Master's Degree Thesis in Telecommunications Engineering

Low Complexity Schemes for High-Performance MIMO Transmission in Quasi-Static Fading Channels

Lamarca Orozco, Meritxell
Thesis' Director

Mateu Mercader, Pedro
Author

Departament de Teoria del Senyal i Comunicacions (TSC)
Universitat Politècnica de Catalunya

18/10/2016

Low Complexity Schemes for High-Performance MIMO Transmission in Quasi-Static Fading Channels

Abstract

Background In the last years Multiple-Input Multiple-Output (MIMO) communications technology has become one of the leading techniques to attain increased data throughput alongside higher reliability and robustness of transmission. It provides substantial gains over single input and single output transmission schemes at the cost of increased physical and computational complexity. One of the biggest research fields in the subject has been studying ways to reduce the aforementioned computational complexity while still having noticeable gains over SISO channels.

Several methods have been proposed with that goal in mind. In the case of the quasi-static fading channel, however, most of them rely upon inter-block temporal diversity, such as D-BLAST and Threaded Spacetime Techniques (TST). These kinds of solutions have they own drawbacks themselves, such as initial phases in which the results are suboptimal, increased delays over those caused by state-of-the-art the error correcting algorithms such as LDPC, or the need to use specific receivers that need to use demapping to decoder iterations.

Thesis Scope This master's degree thesis focuses on the steps followed, and the results attained, of the research of a computationally simple method to get as close as possible to the outage capacity of a quasi-static fading channel with AWGN noise, where the receiver knows the channel but the transmitter does not. In order to do so, the desired results of this investigation would be to find an algorithm that:

- Can be quickly escalated to an arbitrary amount of MIMO antennas and bits transmitted per antenna, without falling into the exponential increases of complexity typically needed.
- Has a decoding stage that is based on basic arithmetic operations, which could be translated into the factor-graphs needed for a proper *soft-based* decoding of the received symbols.
- Has non-complex encoding stages.
- Doesn't incur in additional delays, nor has suboptimal initialization phases.
- It should be able to get as close as possible to the outage capacity of the channel.

In order to reach such milestones, this work draws upon Lamarca's 2006 "Random Labeling: A New Approach to Achieve Capacity in MIMO Quasi-Static Fading Channels" paper, in which a novel approach to achieve the outage capacity of a quasi-static fading was presented. While capacity-wise flawless, the method was at this stage a theoretical construct, as its complexity growed exponentially with the amount of antennas and bits per antenna used.

Thesis Results As will be shown in this document, the random encoding and decoding state developed in this thesis meets positively all of the goals set for it, providing a system to reach close to optimal outage capacities while at the same time being intrinsically fast to escalate to larger MIMO configurations or transmit constellations. As an extra benefit, the method requires error correction rates which can be easily be reached by modern parity check or convolutional codes. On the other hand, the Binary MMSE Demapping developed for this system provides a very simple and affordable demapping device, which in the future could be further improved in terms of spectral efficiency if iterations between the demapper and the decoder are to be allowed.

Collaborations

Departament de Teoria del Senyal i Comunicacions (TSC)

Contents

1 Introduction

1.1 Thesis Context

The almost ubiquitous use of multiple-input multiple-output(MIMO) transmissions schemes in nowadays high-performance communication devices is a testament to its ability to reach greater levels of performance and reliability over traditional single-input single-output(SISO) systems.

In order to achieve such gains, MIMO systems rely on multipath propagation of the transmitted signals, aiming to increase both the reliability of the link and its spectral efficiency. A huge research effort by the international community has been carried out the last 15 years in order to improve both parameters through new, improved methods.

One of most important phenomenons that the researchers had to counteract in order to improve the efficiency of MIMO transmissions was the fading. As will be seen with greater detail in 2.2, fading causes attenuations in the signal that follow certain statistical properties, such as Rayleigh or Rician distributions. In some cases, fading varies in time slowly enough to be approximated to be constant for the whole duration of the transmission of a error-correcting encoder datablock. In such cases, it's said that we are in a situation of *quasi static fading*.

Our thesis is designed to work in such *quasi static fading* case, alongside i.i.d AWGN noise and the fact that the receiver knows the channel but the transmitter does not (it has no channel state information). One of the biggest problems caused by slow fading is that there is no deterministic way to ensure that all transmitted datablocks are decoded properly at the receiver. For a given rate r, we can only set a probability that the received datablocks will be correctly decoded for a specific percentage of the time. This is called the *outage probability*. The capacity of the channel itself will be measured in terms of this outage probability; this is the *outage capacity*, and as mentioned, only makes sense when it's related to a specific percentage of time in which the channel is allowed to be in outage. Technically speaking, the capacity of any channel that is in outage is 0 [2]. Both of these aspects are seen more in-depth in 1.

On the other hand, in the case of multilevel coding systems (MLC), another undesirable effect is that for every new channel realization, the attenuation caused by the fading is different and the distortion it causes in the signal affects unevenly the information carried by each transmitted bit [2]. This means that, even for two channels with the same global outage capacity, without countermeasures the rates required for the error-correcting stages of a single bit (single MLC level) vary widely depending on the channel itself. Hence if we want a small outage probability, we have to set error correcting rates that cover some of the worst case scenarios, and therefore a lot of spectral efficiency is lost trying to counteract the fading effects.

Several methods have been proposed that can counteract this effect. Some of the most well known ones, such as D-BLAST [2][4], reduces its impact by performing a temporal

interleaving of information, which spans several error-correcting encoder datablocks, so that the transmission itself is spreaded through various channels and antennas (thereby exploiting spatial diversity). The problem is that this system introduces a starting phase in which the performance of the system is very suboptimal, and also creates much larger delays than those strictly necessary for error-correction. Furthermore, the structures it uses are prone to error propagation. Other methods, like Threaded-SpaceTime (TST) architectures [5], are based on genie receivers, which intrinsically need for iterations between the demapper and the decoder stages.

In order to counter the drawbacks of the systems present at the time, Lamarca's [1] paper proposed a novel method of reaching the outage capacity in quasi-static fading channels. It had the desirable properties of reaching such levels without needing iterations between the demapper and the decoder, without starting phases and without introducing additional delays to the error-correcting decoder stages.

The key to that solution was to map every possible original binary vector to be transmitted, $\mathbf{b}$, to a randomly generated vector $\mathbf{c}$; the vector that was finally transmitted was the latter. This mapping was obviously done taking into account that $\mathbf{c}$ had to be invertible, and the operation could be reversed without losses. The solution performed very well in all of the aspects it tried to address.

This method was the starting point of this thesis. The main limitation of the method proposed on the paper, as it was, is that it was impossible to scale to large amounts of simultaneously transmitted bits. This was due to two reasons. Firstly, it depended on lock-up tables to perform the random encoding and decoding. Secondly, the only demapper that would work at that point would be a maximum a posteriori demapper (MAP), whose complexity is know to grow at a 2^{N_b} factor, with N_b being the number of simultaneously transmitted bits.

Therefore, new methods where needed to be able to do this random encoding and decoding processes for an arbitrary amount of bits, with a low cost in terms of computational performance. Furthermore, a new demapping scheme would have to be devised so that we're not limited to smaller N_b values. This is what was the starting point of the objectives of the present thesis, which are detailed in the next section.

1.2 Objectives

Firstly, this section will set a series of global restrictions that will delimitate the environment in which this thesis was developed. These global restrictions can be succinctly put as:

Restrictions

- The channel has independent, identically distributed AWGN noise.

- The fading in the channel is constant for the whole duration of the error-correcting datablock transmission, which means that it's *quasi-static fading*.
- The receiver knows the channel but the transmitter does not, that is, it doesn't have channel state information (csi).

Secondly, as briefly summarized in the document's abstract, the objectives of this thesis can be, at the highest level, simplified to the following concepts:

- Finding a MIMO system that can be quickly escalated to an arbitrary amount of MIMO antennas and bits transmitted per antenna, without falling into the exponential increases of complexity typically needed.
- Such a system has a decoding stage that is based on basic arithmetic operations, which could be translated into the factor-graphs needed for a proper *soft-based* decoding of the received symbols.
- Has non-complex encoding stages.
- Doesn't incur in additional delays, nor has suboptimal initialization phases.
- It should be able to get as close as possible to the outage capacity of the channel.

In order to achieve these objectives, in first place it is necessary to find and replace any system stage whose computation time is exponential. In the case of the random decoding proposed on Lamarca's paper [1], which marked the foundations for the research done in this thesis, there are two obvious exponential time stages in use:

- **Random Decoding Stage:** As defined in the paper, and explained in Lamarca's paper [1], the random decoding stage is executed by direct mapping between the original set of words and the randomized set of words. The application used in this mapping is obviously invertible, which means that both sets include the full amount of possible binary words of a given size. This method requires storing a table of size 2^{N_b}, where N_b is the number of bits used.
- **Demapping Stage:** The demapper used in the paper was the optimal one, that is, Maximum a Posteriori demapping (MAP). As will be seen in detail in 2.3, this demapping method requires evaluation of either the probabilities of transmission, or likelihoods, of each of the words that were possibly transmitted, which once again brings us to computing needs proportional to 2^{N_b} operations per demapping stage.

Including these two requirements, and adding the fact that both software implementation and thesis documentation will be needed to prove the researched methods and document them, the objectives of this thesis can be more specifically stated as shown in the following table.

Thesis Objectives

1. Develop a new **random encoding and decoding stage**, with the following features:
   - Both stages must be computationally simple, allowing non-exponential time calculations.
   - Both stages must be easy to design for a trivial amount of bits. This implies that good encoding or decoding stages shouldn't be very difficult to find, and specially, that given a particular encoder or decoder stage, it's specific counterpart (decoder/encoder respectively) is easy to design.
   - The decoding stage must be able to receive as input *soft* values (probabilities of each bit rather than deterministic values). This stage must also be able to feed *soft* output values to the posterior stages.
   - Once used in the system, this stage must allow to get outage capacities close to the theoretical maximum ones without introducing extra delays. The first requirement obviously means that the decoding stage must be the exact inverse of the encoding stage. Therefore under ideal circumstances the output word of the decoding stage would be exactly the same than the word sent to the encoding stage.
2. Develop a new **mapping and demapping stage**, which fulfills the following criteria:
   - Both stages must be computationally simple, allowing once again execution in non-exponential time for a large number of bits per MIMO channel use.
   - Both stages must be easy to design for a trivial amount of bits and antennas.
   - The demapping stages must be able to feed *soft* output values to the posterior stages (random decoding stage in this case).
   - This stage must be able to attain reasonable performances in spectral efficiency, without introducing extra delays over the MAP demapper.
3. Create a **software implementation** of the complete MIMO transmitter and receiver, including the final solutions for both stages as well as the most important previously researched ones. These implementations will also have ideal versions of each of the stages, thereby providing a reference to calculate the capacity losses created by each stage. Use this implementation to simulate the results obtained in different ranges of SNRs, for each subtype of stage.
4. Create the **thesis documentation** itself, covering all of the topics summarized in the Thesis Structure section 1.3.

1.3 Thesis Structure

Each section of this document addresses one specific part of the objectives that were detailed in 1.2. The thesis is structured in the following chapters:

1. Introduction: This chapter itself. The introduction summarizes the context of project, shows the objectives that were set up for it, and, in this subsection, includes an explanation of the structure devised for the document.

2. Description of the Researched System: The chapter focuses on both the theory behind the developed system, as well as an explanation of which methods have been used in the system, including novel ones (introduced in this document), researched but discarded ones and previously existing ones. The chapter begins with an overview of the architecture that has been used for the whole system, depicting each one of the stages of it, and subsequently starts explaining each single stage individually.

Typically, each subsection starts with an introduction, in which the theoretical framework of the stage is briefly explained, and then focuses on the stage element themselves. Generally, theoretical explanations are given only for either the most important concepts or those rarely if ever seen during the master's degree lectures.

On the other hand, in the case of the novel stage of "Random Encoding/Decoding", a subsection is devoted to show the most important intermediate devised designs that were finally discarded, and explains the reasons of why they were superseded by other newer designs.

3. Results: The section firstly introduces some mathematical concepts needed to explain how the system metrics were evaluated. After this, it shows up the most significant tests done using the implemented software simulator, thereby proving the spectral efficiency, resulting rate dispersion and rate per bit needed, and other metrics useful to prove the results of the devised system.

4. Final Proposed System: In this chapter, taking as reference the results of the previous chapter, a final proposed system is briefly described taking the element described in the "Description of the Researched System" which has the best overall performance.

5. Software Implementation: Focuses on explaining the software programs that were implemented in this thesis in order to develop, and test the system, as well as explaining how to operate the most useful ones.

6. Conclusion: A summary of the research process carried in this thesis and the most relevant information gathered, as well as the non-tested possibilities which could potentially yield improvements in the system and that would be interesting to check in the future.

7. Annexes: Includes extended information about discarded novel stages, which were at one point of the thesis the proposed one, and which although superseded by newer versions provides a theoretical framework that may be useful for other situations.

2 Description of the Researched System

2.1 General Architecture Overview

2.1.1 Introduction

At the highest level, the architecture of a generic MIMO transmitter and receptor system can be summarized as shown in figure 1.

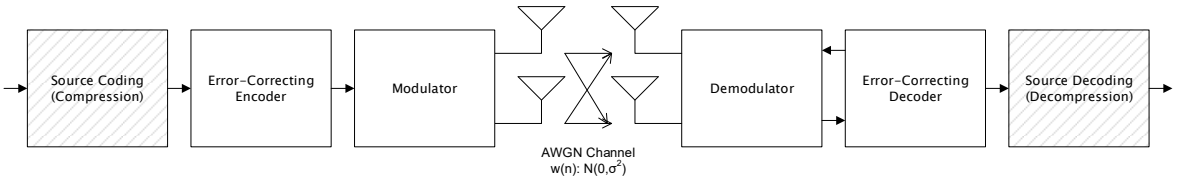

Figure 1: High Level MIMO Transmitter Receiver Schematic

At this level, both the transmitter and receiver have the following stages:

- **Source Coding and Decoding:** In the transmitter, this stage is responsible for the compression of the original data. This stage outputs a vector of bits whose 0 and 1 probability is approximately the same ($P(b = 1) = P(b = 0) = 0.5$), which implies that the vector's entropy is maximized and ideally close to 1. This fact is very important for the following stages, as they all assume that the *a priori* probabilities of 0 and 1 are equal. In the receiver, the bits are decompressed and restored to their original form. In any case, **this stage is out of the scope of this thesis, and from this point onwards it is merely replaced by the assumption that the initial bits fulfill the aforementioned probabilities**.

- **Error-Correcting Encoder and Decoder:** The purpose of these stages is adding redundancy to the sequence to be set so that upon decoding error correction routines allow full recovery of the sent data. Any code whose efficiency allows performances approaching channel's capacity can be used in this stage. Information about this stage is extended in subsection 2.5.

- **Modulation and Demodulation Stages:** In the transmitter, this stage is responsible for the generation of the transmitted symbols for each antennas. In the receiver, it receives the signals through the receiver's antennas, and processes the signals so that the probabilities for each bit output by the Error-Correcting Encoder is calculated and send to the Error-Correcting Decoder. This stage is extended in the next subsections 2.1.2 and 2.1.3.

- **AWGN Channel:** The radio-wave medium will be characterized as an additive white Gaussian noise channel with quasi-static fading. See 2.2 for further information.

As mentioned above, subsequent, more detailed versions of the architecture merely omit the Source Coding and Decoding stages, and consider that the information send to the transmitter already complies with the fact that 0 and 1 bits should have equal probabilities.

Note that the Demodulation Stage and the Error-Correcting Decoder stage are connected in both directions. This is due to two distinct factors. In the first place, capacity achieving receivers make use of *soft* information at the demodulator and error-correcting stages, that is, they use probabilities rather than deterministic or *hard* binary decisions. On the other hand using this *soft* information allows for iteration between stages, which, as will be seen in later chapters, improve the overall results of the system.

2.1.2 Transmitter Architecture

This section will focus on providing a detailed description of the transmitter that will actually be used in our system. The underlying scheme of the proposed method is based upon already known multistage decoding MIMO communication systems, namely, a **Multilevel Coding (MLC)** setup with random labeling.

Let N_b be the number of bits that the system can transmit simultaneously through all antennas. On the transmitter, MLC designs divide the original bitstream to be transmitted into N_b unevenly sized layers, each of which processed by a error-correcting code of a different rate (see 2.5 for further details). On the receiver, MLC designs will demap a single layer, decode it and send *a posteriori* information back to the demapping stage in order to process the next layer, until all of them have been decoded. This method has two important advantages:

1. It's capable of reaching the capacity for any constellation labeling used.
2. For any single layer, it does not require iterations between the demapper and the error-correcting decoder.

The proposed transmitter only requires two key modifications of generic MLC designs: the addition of a pseudo-random labeling method and its subsequent labeling decoding stages. The new stages, based on factor-graph marginalization, replace the methods proposed in [1].

The transmitter, illustrated in figure (2), is responsible for both treating the data to be sent and finally emitting it. In order to properly explain how the transmitter works on a global scale, this section will be divided in two separate topics:

- **Stage Summary**: Which will briefly describe each state purpose, while at the same time describing its main inputs and outputs.
- **Transmitter Data Flow**: Will summarize the inner workings of the transmitter, focusing mostly on the flow of data between subsystems and order-of-execution of the different stages.

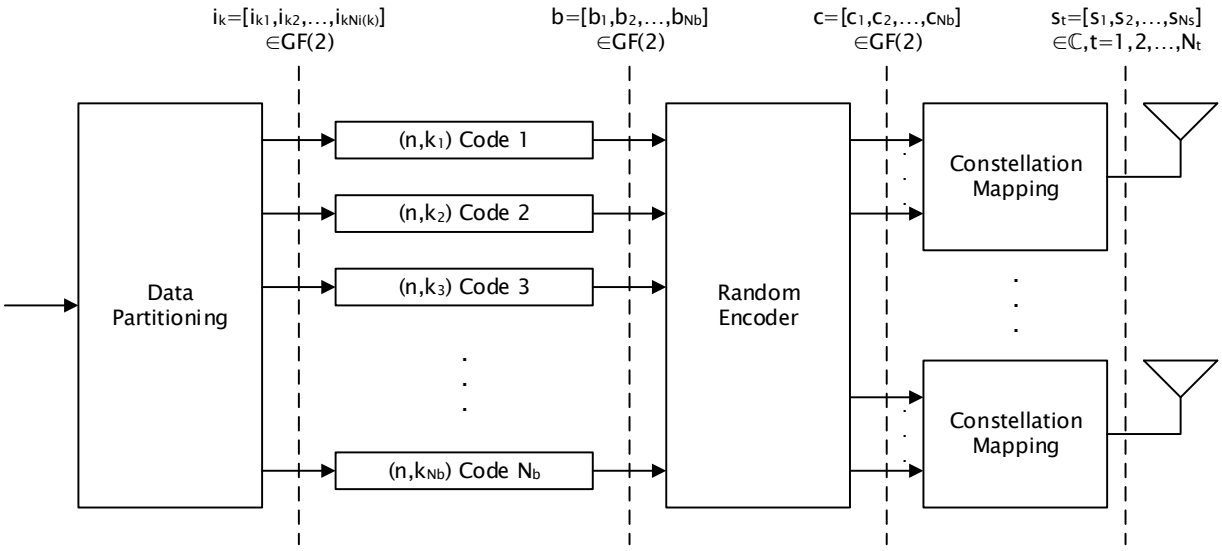

Figure 2: Transmitter Schematic

Stage Summary

Let N_d be the total number of source bits transmitted in one transmission *slot*. Given that the proposed method makes use of block error-correcting codes, the size of the slot can be defined as $\sum_{\forall i} k_i$, where k_i is the input vector size used at each *i-th* MLC layer (n, k_i) code. As depicted in figure (2), the transmitter can be seen as a four stage system:

- **Data Partitioning**: This stage receives the data to be sent, $\mathbf{d} = [d_1\ d_2\ \dots\ d_{N_d}]$, $d \in GF(2)$, and partitions it in a set of unevenly sized vectors, $\mathbf{k}_i$, $i = 1, 2, \dots, N_b$ being the total number of MLC layers used. This partitioning is needed due to the unequal channel capacity of each transmitted bit within a symbol. In compliance to the MLC scheme, each bit will be treated by a code with a specific rate for optimal decoding, allowing efficient decoding for any type of constellation labeling [3]. This custom rate for each bit means that the amount of useful, non-redundant data contained inside each bit will be different, hence the data to be handled to the codes from the data partitioning stage will have to be of a specific size for each code.

- **Error Correcting Codes**: In each transmitter there will be N_b different codes, each with its own rate $r_i = k_i/n$ with $i = 1, 2, \dots, N_b$. The purpose of these stages is adding redundancy to the sequence to be set so that upon decoding error correction routines allow full recovery of the sent data. Each code will have a $1 \times k_i$ sized $\mathbf{k}_i$ vector as its input, and will output a $1 \times n$ sized $\mathbf{b}_i$ vector.

- **Random Coding Stage**: In this stage the $\mathbf{b} = [b_1\ b_2\ b_3\ \dots\ b_{N_b}]$ input bits are converted to $\mathbf{c} = [c_1\ c_2\ c_3\ \dots\ c_{N_b}]$. This stage correlates the inputs $\mathbf{b} \in GF(2)^{1\times N_b}$ to form the outputs in $\mathbf{c} \in GF(2)^{1\times N_b}$, given that, as will be seen, such correlation allows a capacity achieving transmission through a quasi-static fading MIMO channel.

- **Constellation Mapping Stages**: Each vector $\mathbf{c}$ is subdivided in N_t $\mathbf{c}_v$ vectors, with

$v = 1, 2, \ldots, N_t$. N_t is the number of transmitting antennas. Every $GF(2)^{1\times N_b/N_t}$ vector $\mathbf{c}_v$ is mapped to a single complex symbol $s \in \mathbb{C}$, in accordance to the constellation and labeling assigned to each mapper. The symbol's modulus and phase are the amplitude and phase of the signal that is finally transmitted through the antenna.

Transmitter Data Flow

A top-level, summarized version of the process of transmitting a set of symbols for an entire stream of data $\mathbf{d}$ (considering the initial time interval $t = 0$ and the final time interval of the timeslot $t = n$) can be described as follows:

1. Firstly the Data Partitioning stage receives the entire stream of data $\mathbf{d}$ and splits it into the unevenly sized set of vectors $\mathbf{k}_i$ sending them immediately to the N_b *Error Correcting Codes* stages.

2. The N_b Error Correcting Encoding stages receive the set of uneven vectors $\mathbf{k}_i$ and calculate the set of equally sized $\mathbf{b}_i$ vectors. However, the entire set of vectors $\mathbf{b}_i$ isn't sent immediately to the next stage. Each instant, a subset $\mathbf{b}$ of the whole group of $\mathbf{b}_i$ vectors is forwarded to the random encoder. The vector $\mathbf{b}$ sent in the instant t is the t-th column of the matrix $\mathbf{B}$ that results from using each vector $\mathbf{b}_i$ as the i-th row of the aforementioned matrix:

$$\mathbf{B} = \begin{bmatrix} \mathbf{b}_1 \\ \mathbf{b}_2 \\ \vdots \\ \mathbf{b}_i \\ \vdots \\ \mathbf{b}_{N_b} \end{bmatrix} = \begin{bmatrix} b_{11} & b_{12} & \cdots & b_{1n} \\ b_{21} & b_{22} & \cdots & b_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ b_{i1} & b_{i2} & \cdots & b_{in} \\ \vdots & \vdots & \ddots & \vdots \\ b_{N_b1} & b_{N_b2} & \cdots & b_{N_bn} \end{bmatrix} \tag{1}$$

$$\begin{array}{ccccc} t=1 & & t=2 & \cdots & t=n \\ \mathbf{b} = [b_{11}\ b_{21}\ \ldots\ b_{N_b1}\ ] & & \mathbf{b} = [b_{12}\ b_{22}\ \ldots\ b_{N_b2}\ ] & \cdots & \mathbf{b} = [b_{1n}\ b_{2n}\ \ldots\ b_{N_bn}\ ] \end{array} \tag{2}$$

3. For each t until $t = n$:

   (a) As stated above $\mathbf{b}$ is equal to the t-eth column of $\mathbf{B}$. The Random Coding stage process the vector $\mathbf{b}$ and outputs $\mathbf{c}$. Each vector $\mathbf{c}$ is subdivided in N_t $\mathbf{c}_v$ vectors, with $v = 1, 2, \ldots, N_t$. N_t is the number of transmitting antennas. Every $GF(2)^{1\times N_b/N_t}$ vector $\mathbf{c}_v$ is sent to the Constellation Mapping stage of each antenna.

   (b) The Constellation Mapping stages each convert its input $\mathbf{c}_v$ vector into a complex symbol that is modulated and transmitted through radio waves.

Of course, for each subsequent data stream $\mathbf{d}$ to be emitted the process described would be repeated.

2.1.3 Receiver Architecture

As in the case of the transmitter, the scheme of the receiver is based upon multilevel coding (MLC) setup for MIMO, with the novel addition of random labeling decoding. The receiver itself is, at first glance, a reverse-order set of the (inverse) stages found in the transmitter, as shown in picture (3). However, upon close examination, a couple of relevant differences arise between both systems:

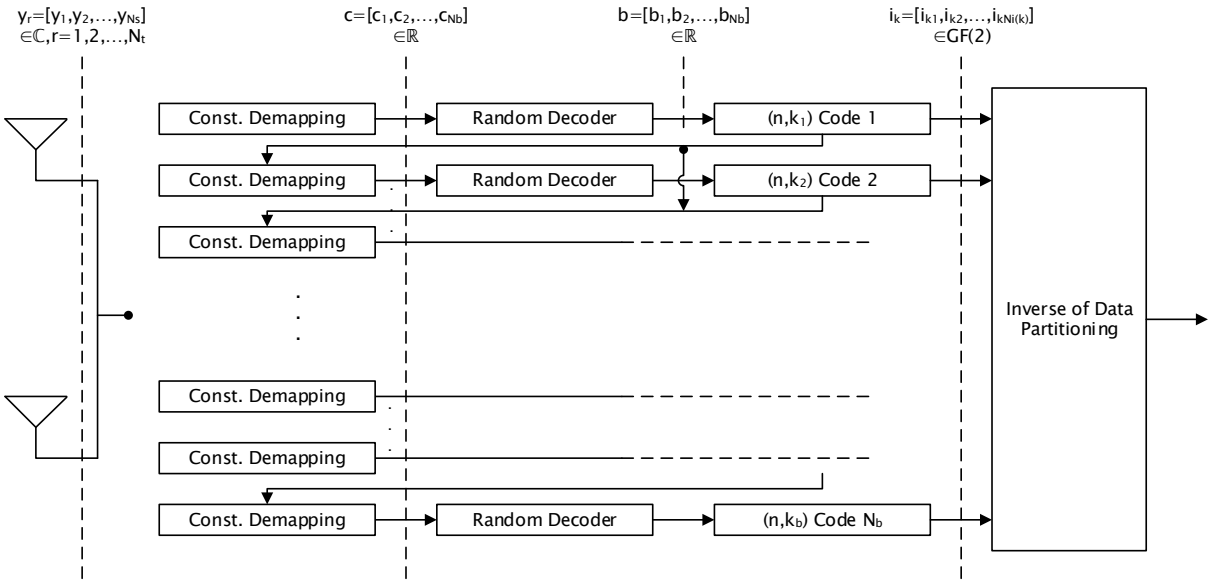

Figure 3: Receiver Schematic

- Unlike the deterministic nature of the transmitter, in order to attain maximum capacity schemes the receiver has to work in probabilistic terms, being, for the most part, a *soft* system. In order to allow for calculations in probabilistic terms, and as seen in illustration (2), the decoder uses real vectors instead of binary ones.
- As usual in the MLC setup, and unlike in the transmitter, the information obtained after each (n, k_i) code, $i = 1, 2, \ldots, N_b$, is fed to all subsequent stages (n, k_j), $j > i$.

The proposed implementation makes heavy use of the Log Likelihood Ratio(LLR), llr from now onwards, as a mean to transmit and process the probabilities involved in the receiver. This log likelihood ratio is a mathematical tool that significantly simplifies some of the gates and stages present in this scheme. The equation (3) shows the relationship between the probabilities of a binary event A and its log likelihood ratio.

$$\begin{aligned} a &\in \{0,1\} \subseteq GF(2) \\ \mathrm{P}\left(A=a\right) &\in (0,1] \subset \mathbb{R} \\ \mathrm{LLR}\left(A\right) &= \ln\left(\frac{\mathrm{P}(A=1)}{\mathrm{P}(A=0)}\right) \end{aligned} \tag{3}$$

In order to properly explain how the receiver works on a global scale, this overview will be divided in two separate topics:

- **Stage Summary**: This section will briefly describe each state purpose, while at the same time describing mathematically its main inputs and outputs. Inter-stage backwards message passing will not be treated in this section.
- **Receiver Data Flow**: Will summarize the inner workings of the decoder, focusing mostly on the flow of data between subsystems and order-of-execution of the different stages.

Stage Summary

The receiver is composed of the following stages:

- **Constellation Demapping Stages**: The vector $\mathbf{y} = [y_1\ y_2\ \dots\ y_{N_r}] \in \mathbb{C}^{1\times N_r}$ (N_r being the number of receiving antennas), containing each received signal y_i, is processed into a vector of probabilities of every bit of the codified transmitted symbol. This demapping stage should be implemented as an APP module: each bit *extrinsec* probability is calculated using the *a priori* information obtained while decoding previous bits. The output vector $\mathbf{c} = [c_1\ c_2\ \dots\ c_{N_b}] \in \mathbb{R}^{1\times N_b}$ contains the aforementioned probabilities, in the form of log likelihood ratios. Example (4) shows llr of a given $\mathbf{c}$ after b_1 and b_2 have been decoded.

$$\begin{aligned} c_1 &= \ln\left(\frac{\mathrm{P}(c_1=1/\mathbf{y},b_1,b_2)}{\mathrm{P}(c_1=0/\mathbf{y},b_1,b_2)}\right) \\ c_2 &= \ln\left(\frac{\mathrm{P}(c_2=1/\mathbf{y},b_1,b_2)}{\mathrm{P}(c_2=0/\mathbf{y},b_1,b_2)}\right) \\ \vdots &= \qquad\vdots \\ c_{N_b} &= \ln\left(\frac{\mathrm{P}(c_{N_b}=1/\mathbf{y},b_1,b_2)}{\mathrm{P}(c_{N_b}=0/\mathbf{y},b_1,b_2)}\right) \end{aligned} \tag{4}$$

- **Random Decoding Stage**: This stage decodes the llr probabilities fed by the demapper, $\mathbf{c} = [c_1\ c_2\ c_3\ \dots\ c_{N_b}] \in \mathbb{R}^{1\times N_b}$, and as main output will calculate the vector of llr $\mathbf{b} = [b_1\ b_2\ b_3\ \dots\ b_{N_b}] \in \mathbb{R}^{1\times N_b}$. It's basically a *soft* inverse of the random encoding stage. As will be seen in chapter 2.4, this stage won't be equal for the whole timeslot.
- **Error Correcting Codes**: In each receiver there will be N_b different codes, each with its own rate $r_w = k_w/n$ with $w = 1, 2, \dots, N_b$. The purpose of these stages is remove the redundancy added in the transmitter to the sequence, hopefully fixing in the process any error during transmission. A *soft* code that inverses the process used in the transmitter must be used in this stage. Each code will have a $\mathbb{R}^{1\times n}$ sized vector $\mathbf{b}_i$ as its input vector, whose contents will be the set of llr fed by the random decoding stage. The stage will have as final output a $GF(2)^{1\times k_i}$ sized vector $\mathbf{k}_i$.
- **Data Partitioning**: Finally, this stage assembles the sequence originally split by the emitter into the sent stream of data. This element receives the unevenly sized input vectors, $\mathbf{k}_i$ with $i = 1, 2, \dots, N_b$, and outputs the estimated data that was originally sent, $\mathbf{d} = [d_1\ d_2\ \dots\ d_{N_d}] \in GF(2)^{1\times N_d}$. Once again, $\sum_{\forall i} k_i = N_d$. As in the case of

the emitter, this partitioning was needed due to the unequal channel capacity of each transmitted bit within a symbol. In compliance to the MLC scheme, each bit will be treated by a code with a specific rate for optimal decoding, allowing efficient decoding for any type of constellation labeling. This custom rate for each bit means that the amount of useful, non-redundant data contained inside each bit will be different, hence the data handled by the codes to the data partitioning stage is of a specific and unique size for each code.

Receiver Data Flow

A top-level, summarized version of the process of receiving and processing a set of symbols for an entire stream of data $\mathbf{d}$, considering the initial time interval $t = 0$ and the final time interval of the present timeslot $t = n$, can be described as follows:

1. For t=1 to t=n
   (a) Receive the symbol y_i in each of the $i = 1, 2, ..., N_r$ receiving antennas corresponding to the current time interval t. The vector $\mathbf{y}$ stores each y_i symbol. The t-eth column of $\mathbf{Y}$ is equal to the currently generated $\mathbf{y}$ symbol.
2. For Bit=1 to Bit=N_b
   (a) For Cont=1 to Cont=n
      i. Process the set of symbols of the *Cont*-th column of $\mathbf{Y}$ in the MIMO *Constellation Demapping* stage, using the latest *a priori* information if it has already been calculated ($\mathbf{Y}_{apriori}$), until the current $\mathbf{c}$ is obtained.
      ii. The *Random Decoding* stage processes the vector $\mathbf{c}$ belonging to the current *Cont* set and outputs $\mathbf{b}$.
      iii. The *Cont*-th column of the matrix $\mathbf{B}$ acquires the value of the current vector $\mathbf{b}$.
   (b) The *Error Correction Code* stage of the level 'Bit' processes the Bit-th row of matrix $\mathbf{B}$. The vector $\mathbf{k}_i$ with $i = Bit$ is its main forward output. As backwards output, with the redundancy introduced by the *Error Correction Code* in the transmitter it calculates and sends backwards *a priori* information in the form of the matrix $\mathbf{B}_{apriori}$.
   (c) For Cont=1 to Cont=n
      i. The *Cont*-th column of the matrix $\mathbf{B}_{apriori}$ is used as by the Random Decoding stage backwards inputs, generating the output $\mathbf{c}_{apriori}$.
      ii. $\mathbf{c}_{apriori}$ is used by the *Constellation Demapping* stage as backwards input, it uses this *a priori* information to process (and update) matrix $\mathbf{Y}_{apriori}$, being the *Cont*-th column of that matrix the one that is modified at each step.
3. Data Partitioning stage combines all existing $\mathbf{k}_i$ vectors and outputs the datastream $\mathbf{d}$.

Note that 'Bit' and 'Cont' are internal variables used to describe the data flow. N_b is, as usual, the number of bits to be processed (as it's equivalent to the number of levels in this proposed MLC scheme), while the expressions 'forward' and 'backward' make reference to the direction of the flow of data: forward when the information is sent towards the final decoded datastream **d**, and backwards in the opposite direction.

It's worth noticing that while the step 1, the detection of magnitude and phase of the symbols received at each time interval t, has to be done in realtime, the rest of steps can be processed later, in background.

2.2 Channel Characterization for MIMO Systems

It's possible to model a transmission channel used by a multiple-input multiple-output system as an extension to the one in use for single-input single-output systems. Equation 5 and 6 are the most common expression that relates the output signals of the transmitter with the input signals of the receiver.

$$\begin{bmatrix} y(t)_1 \\ y(t)_2 \\ \vdots \\ y(t)_{N_r} \end{bmatrix} = \begin{bmatrix} h(t)_{00} & h(t)_{01} & \dots & h(t)_{0Nt} \\ h(t)_{10} & h(t)_{11} & \dots & h(t)_{1Nt} \\ \vdots & \vdots & \ddots & \vdots \\ h(t)_{Nr0} & h(t)_{Nr1} & \dots & h(t)_{NrNt} \end{bmatrix} \begin{bmatrix} s(t)_0 \\ s(t)_1 \\ \vdots \\ s(t)_{N_t} \end{bmatrix} + \begin{bmatrix} n(t)_0 \\ n(t)_1 \\ \vdots \\ n(t)_{N_r} \end{bmatrix} \tag{5}$$

$$\mathbf{y}(t) = \mathbf{H}(t)\mathbf{s}(t) + \mathbf{n}(t) \tag{6}$$

N_t is the number of antennas of the transmitter, N_r is the number of antennas of the receiver, $\mathbf{H}(t)$ is the channel matrix, $\mathbf{s}(t)$ the signal sent from each transmitter antena, $\mathbf{n}(t)$ is the noise, and finally $\mathbf{y}(t)$ is the received signal at each of the receiver antennas.

The received vector $\mathbf{y}(t)$ is the addition of the transmitted signals of each antenna, modified in phase and amplitude by the channel matrix elements, plus the noise present in the system. Therefore there will be three elements that affect the transmission of our signals in any MIMO transmission:

1. The noise of the channel, that alongside the signal power will determine the SNR and therefore the maximum attainable capacity for a given MIMO system.

2. The fading of the channel, defined as the change of magnitude of the signal caused by the channel matrix **H**, which will also impose limitations in the bandwidth available.

3. The interference between the transmit antenna signals, given that the channel matrix **H**, multiplies all of the transmitted antenna signal and therefore each received signal y_i has components of all of the transmitted signals.

Note that under perfect circumstances, **n=0** and **H=I**. This would imply that the channel has no noise, there is no fading as the magnitude of $h_{ii}(t) = 1$ and there is no interference as for any (i, j) $j <> i, h_{ij}(t) = 0$. Obviously, this situation is far from realistic.

Regarding the nature of the noise, **this thesis always considers that the noise of the channel is always independent and identically distributed Additive White Gaussian Noise (AWGN), with zero mean and variance** σ^2. This implies that the noise has the same power across all the frequencies, and that its probabilistic nature is that of a gaussian distribution, specifically one with zero mean and variance σ^2 ($\mathcal{N}(0, \sigma^2)$). Furthermore, its independent and identically distributed nature means that many statistical operators that will be used in the system's demapping stage will be greatly simplified.

On the other hand, the nature of the fading during a given time period sets a set of distinct types of fading:

- **Slow Fading Channel:** the channel can be approximated as one that doesn't change for long periods of time, during which the matrix **H** can be considered constant. When the channel can be approximated as one that doesn't change for the duration of the timeslot, that is, the transmission of a single set of bits whose size is the block size of the error-correcting stages, it's typically called *quasi-static fading channel* or *block fading channel*. Between different blocks of bits, the channel changes freely following different statistical distributions.

- **Fast Fading Channel:** the channel changes continuously, the matrix is different for each single transmission.

Note that while in the case of the Fast Fading Channel, the expression 6 cannot be simplified, for Slow-Fading and Quasi-static Fading channels, during the duration of a single error-correcting block transmission, the equation can be simplified as shown in 7. The **H** matrix will remain constant in that situation.

$$\mathbf{y}(t) = \mathbf{Hs}(t) + \mathbf{n}(t) \tag{7}$$

The statistical distribution of the fading can be varied. Two of the most common statistical distributions are the Rayleigh and Rician one. Both distributions model the attenuation caused by the interference of a signal by itself, due to multipath propagation. Rician distribution is common in environments in which there is a strong line-of-sight signal that overshadows the rest of the signals. On the other hand, Rayleigh distribution is used in environments in which there is no clear line-of-sight, and therefore the fading itself is flat.

The thesis is developed under the assumption that the channel can be modeled as a Quasi-Static Rayleigh Fading Channel. It's worth noting that while the system should be able to operate under other statistical distributions, it would not work for the case of a fast-fading system.

2.3 Mapping and Demapping Stage

2.3.1 Introduction

The mapping stage is the one responsible for the conversion of the received bits into a signal with a given amplitude and phase, mathematically represented by a complex number. In the case of MIMO systems, there is a mapping stage for each of the N_t transmitter antennas.

The demapping stage receives the signal in the form of an amplitude and a phase, and as our system is based on *soft* information processing, it calculates the probabilities of the received bits. Unlike in the demapper, optimal demapping stages in MIMO system require joint demapping. This means that the demapping of the symbols received in all antennas must be processed together, as will be seen in 2.3.4.

The specific naming and message passing used in this thesis can be seen in detail in the Architecture sections 2.1.3 and 2.1.2, and is subsequently used in the MAP and MMSE demapping described in this thesis.

On these stages, the exact equivalence between a received string of bits and the transmitted complex signal is obtained by using a *constellation map*. This constellation, alongside it's complementary constellation labeling, maps every single string of bits to a single point in the complex plane. Constellation mapping and demapping can be implemented in two ways:

1. As a look-up table: Every single possible string of bits and its assigned point in complex space is stored in memory.
2. As a function: A function exists which, given a specific string of bits, can quickly calculate the assigned point in complex space.

Not all constellations, and constellation labellings can be implemented as a simple function. Note that for the case of look-up table implementations, the memory required increases exponentially with the amount of bits transmitted simultaneously. **Given that one of the objectives of this thesis is to be able to process an arbitrary amount of bits in non-exponential time, it's mandatory to find and use a constellation labeling that can be implemented as a function.** Furthermore, the aforementioned function must be simple enough to be used in the novel demapper that we describe in 2.3.5.

For certain transmitter and receiver architectures, choosing a proper constellation labeling is needed to reach the capacity of the channel. Fortunately, the MLC architecture used in our system is capable of reaching the capacity of the system [3] for any given architecture, as is proven in the Results chapter 1.

As widely know, there are several types of different constellations such as Phase-shift Keying (PSK), in which the points in the complex plane have all the same magnitude but different phase, and Quadrature Amplitude Modulation (QAM), in which such points are placed evenly in the complex field, with changes of both phase and magnitude. **This system**

has been implemented and tested with QAM constellations, specially 16-QAM constellations, although it supports higher order constellations.

2.3.2 Reference Constellations and Constellation Labellings

During the development and implementation of the simulation software, it was decided that one reference constellation (QAM) with two different reference constellation labellings (Grey, SP) should be used with the system to prove that the MLC architecture still complies with one of its properties: achieving capacity for any possible constellation. **Once this point was proven**, as will be seen in the Results section (1), **they were superseded by the ADD constellation labeling** described in 2.3.3.

The first reference constellation was a QAM with **Gray** labeling one. Gray labeling is designed so that the Hamming distance between a binary vector assigned to a given point in the complex space, and another binary vector assigned to an immediately adjacent point of the complex space is always only 1 bit. This type of constellation labellings never induce by themselves losses of capacity, no matter which system architecture is used in the receiver [3].

The second reference constellation was a QAM with **Set Partitioning** labeling one. This constellation is build taking into account that every bit in the $\mathbf{b} = [b_0 b_1 \dots b_{N_b}]$ whose value (0 or 1) is the same, must be in a contiguous area. Typically, at the highest level, b_0, all of the possible strings whose $b_0 = 0$ will be assigned points in the real-negative semi-plane of the complex space, while $b_0 = 1$ will be assigned real-positive points. b_1 points will be divided between imaginary positive and imaginary negative, while subsequent bits will be further divisions of contiguous areas. These constellation labelings typically have losses unless used by specific architectures such as MLC.

2.3.3 ADD Constellation and Constellation Labeling

Introduction

Given the objective of allowing the processing of arbitrary amounts of simultaneously transmitted bits (N_b), as well as the specific needs of the Binary MMSE demapper seen in 2.3.5, Gray or generic Set Partitioning constellation labelings could not be used due to their typical use of look-up tables.

For that reason, ADD constellation labeling was chosen as the most promising one for this thesis. This type of labeling can be seen as a specific type of Set Partitioning labelings, in which there exists a simple mathematical equation that links the binary vector with the complex plane point.

Once it was proven, as shown in 1, that there were no differences in spectral efficiency by using the reference labelings (Gray, SP) or this one, **ADD constellation and labeling scheme became the default one used by the system.**

Definition

As mentioned in the introduction, ADD constellation and constellation labelings are a set of fully square QAM constellations whose main advantage lies on the fact that each one of the constellation's symbols, s, can be directly calculated with linear algebra from the transmitted bits, $\mathbf{c}$. In order to achieve this, each of the transmitted symbols is multiplied by a real or complex factor, α_i or $j\alpha_i$, respectively. The equation (8) illustrates this relationship.

$$s = \begin{bmatrix} \alpha_0 & j\alpha_0 & \alpha_1 & j\alpha_1 & \cdots & j\alpha_{\lfloor N_{ba}/2 \rfloor - 1} \end{bmatrix} \begin{bmatrix} c_0 \\ c_1 \\ c_2 \\ \vdots \\ c_{N_b-1} \end{bmatrix} = \boldsymbol{\alpha c} \tag{8}$$

With N_{ba} being the number of bits transmitted through one antenna and $c \in \{-1, +1\}$. Note that these schemes can be viewed as a summation of nested QPSK signals, as shown in (9).

$$s = \sum_{\forall i} \alpha_i (c_{2i} + j c_{2i+1}) \tag{9}$$

Where $i = 0, 1, 2, ..., \lfloor N_{ba}/2 \rfloor - 1$. Each of the terms of the summation is a point in a QPSK constellation by itself, while the non-normalized transmitted symbol s is the addition of all of the calculated QPSKs points.

In order to achieve QAM constellation schemes, only specific values of α_i can be used. If we want evenly spaced constellation points, such as the typical minimum non-normalized constellation point distance of 2 and 2i in the real and complex axis, respectively, each α_i will be bound to the set of values (10).

$$\begin{aligned} \alpha_0 &= 2^{N_{ba}/2-1} \\ \alpha_1 &= 2^{(N_{ba}-1)/2-1} \\ \alpha_2 &= 2^{(N_{ba}-2)/2-1} \\ &... \\ \alpha_{\lfloor N_{ba}/2 \rfloor - 1} &= 2^0 = 1 \end{aligned} \tag{10}$$

Which can be briefly summarized by the expresion (11).

$$\alpha_i = 2^{(N_{ba}-i)/2-1}, i \in (0, 1, 2, ..., \lfloor N_{ba}/2 \rfloor - 1) \tag{11}$$

Note that this will create a standard, fully square QAM constellation in the case of $N_{ba}/2 \in \mathbb{N}$, such as in the case of 16, 64 or 256 QAM. Otherwise the results would differ from that of the non-square QAM constellations.

In many cases it's convenient to normalize the power of the transmitted signal to a $\sigma_s^2 = 1$. In such cases, a normalizing factor must be included, as shown in (12).

$$\alpha_i(normalized) = \frac{1}{\sqrt{E\{\|s\|^2\}}} \alpha_i(non - normalized) \tag{12}$$

Extension to MIMO systems

In the case of MIMO systems, it's possible to extend the aforementioned equations to provide support for the multiple symbols that are transmitted at once using the N_t transmit antennas. Let $\mathbf{s}$ be the $\mathbb{C}^{1\times N_t}$ vector that stores each of the transmitted symbols on each of the N_t transmit antennas. Let $\boldsymbol{\alpha}$ be a $\mathbb{C}^{1\times(N_{ba}/2-1)}$ vector which contains $[\alpha_0\ j\alpha_0\ \alpha_1\ j\alpha_1 \cdots j\alpha_{N_{ba}/2-1}]$ as values. Finally, let $\mathbf{0}$ ve a $\mathbb{N}^{1\times(N_{bs}/2-1)}$ vector whose values are all 0. The equation (13) shows the relationship between $\mathbf{s}$ and the ADD constellations of each antennas.

$$\begin{bmatrix} s_1 & s_2 & s_3 & \dots & s_{N_t} \end{bmatrix} = \begin{bmatrix} \boldsymbol{\alpha} & \mathbf{0} & \cdots & \mathbf{0} \\ \mathbf{0} & \boldsymbol{\alpha} & \cdots & \mathbf{0} \\ \vdots & \vdots & \ddots & \vdots \\ \mathbf{0} & \mathbf{0} & \cdots & \boldsymbol{\alpha} \end{bmatrix} \begin{bmatrix} c_0 \\ c_1 \\ \vdots \\ c_{N_{ba}} \\ c_{N_{ba}+1} \\ c_{N_{ba}+2} \\ \vdots \\ c_{2N_{ba}} \\ \vdots \\ c_{Nb-N_{ba}+1} \\ c_{Nb-N_{ba}+2} \\ \vdots \\ c_{Nb} \end{bmatrix} \tag{13}$$

Where N_b is the total number of bits sent simultaneously for every channel slot. Note that $\boldsymbol{\alpha}$ matrices could potentially be different for each of the transmitting antennas. However, for the scope of this project $\boldsymbol{\alpha}$ will remain invariant.

Replacing the equation (13) with matrices, the compact result (14) fully describes the ADD constellation and labeling.

$$\mathbf{s} = \mathbf{A}\mathbf{c} \tag{14}$$

Applying the equations in (13) to the channel equation, one can reach the equivalence in (15).

$$\begin{aligned} \mathbf{y} &= \mathbf{H}\mathbf{s} + \mathbf{n} \\ \mathbf{y} &= \mathbf{H}\mathbf{A}\mathbf{c} + \mathbf{n} \end{aligned} \tag{15}$$

The linear relationship in this constellation and labeling scheme between the transmitted symbols $\mathbf{s}$ and the coded bits $\mathbf{c}$ will greatly facilitate the calculation of the low complexity demapper proposed in this document, as shown in 2.3.5. On the other hand, this simplicity will not impact the overall performance of the demapping stage, as MLC schemes such as the one used in this system achieve the capacity for any labelings [3].

This is partially due to the fact these constellations schemes, when used in $N_t \times N_r$ MIMO systems, can be interpreted as equivalent $N_{ba}N_t \times N_r$ systems, in which all of the N_{ba} bits sent in each transmitting antenna is affected by the same fading value h_{ij}. An example of a

2×2 MIMO system with $N_{ba} = 2$ is shown in (16).

$$\begin{bmatrix} y_0 \\ y_1 \end{bmatrix} = \begin{bmatrix} h_{00} & h_{01} \\ h_{10} & h_{11} \end{bmatrix} \begin{bmatrix} \alpha_0 & j\alpha_0 & 0 & 0 \\ 0 & 0 & \alpha_0 & j\alpha_0 \end{bmatrix} \begin{bmatrix} c_0 \\ c_1 \\ c_2 \\ c_3 \end{bmatrix} + \begin{bmatrix} n_0 \\ n_1 \end{bmatrix}$$

$$\begin{bmatrix} y_0 \\ y_1 \end{bmatrix} = \begin{bmatrix} \alpha_0 h_{00} & j\alpha_0 h_{00} & \alpha_0 h_{01} & \alpha_0 h_{01} \\ \alpha_0 h_{10} & j\alpha_0 h_{10} & \alpha_0 h_{11} & \alpha_0 h_{11} \end{bmatrix} \begin{bmatrix} c_0 \\ c_1 \\ c_2 \\ c_3 \end{bmatrix} + \begin{bmatrix} n_0 \\ n_1 \end{bmatrix} \tag{16}$$

$$\mathbf{s} = \mathbf{H}_{eq}\mathbf{c} + \mathbf{n}$$

Thereby proving that a MIMO channel with such ADD constellations and labeling can be understood as an equivalent channel $\mathbf{H}_{eq}$ whose signals are directly the $\mathbf{c} \in \{-1, +1\}$ transmitted bits.

2.3.4 MAP demapper

The *Maximum a Posteriori* (MAP) demapper is capable of translating the received signals phase and amplitude to bit probabilities without information losses. As such, in terms of spectral efficiency, it can be regarded as the reference for any other demapper intended to be tested.

This demapper works by taking into account both the *a priori* information available of the symbols to be demapped, as well as the channel probabilities detected for the symbol in question. The equation 17 shows the process needed to calculate a MAP probability for a given bit c_j.

$$P(c_j = x) = \sum_{\forall t_i : \text{j-th bit } c_j = x} p(t_i) p(\hat{r}/r = t_i) \tag{17}$$

Where t_i is one of the N_s symbols that the transmitter can send, r is the received symbol, and c_j is the bit that is intended to be calculated. As this system uses binary vectors, x will be either $\{0, 1\}$ or $\{1, -1\}$ depending on the interpretation used.

Note that the first time that this demapper is used, there is no *a priori* information of the transmitted symbols, and therefore the term $p(t_i) = 1/N_s$ for any i, where N_s is the total number of symbols. Subsequent calls to this demapper, after the error-correcting codes stages have already processed some data and sent back the *apriori* information of it, will make this term different for each single symbol.

In order to properly use this demapper, it's necessary to avoid using *a priori* information of the bit c_j to be calculated in the right side term of the equation 17. This means that in order to calculate $p(t_i)$, we must set the value of $p(c_j = x)$ to 0,5. As an example, let symbol

t_1 be the one assigned to the binary vector [0 0 0 0]. In order to process c_0, t_1 should be calculated as the expression 18 shows.

$$p(t_1) = 0,5.p(c_1 = 0).p(c_2 = 0).p(c_3 = 0) \tag{18}$$

On the other hand, for MIMO systems, it's also very important to understand that **the symbol t_i is not the same than the symbol sent by each transmitting antenna**, unless the channel matrix $H = I$. This is due to two factors: the presence of fading, which distorts the transmitted symbol in phase and amplitude, and the interference of the symbols sent by the different antennas.

For optimal MAP demapping in such MIMO systems, **joint symbol demapping** must be used. In practice this means that it's necessary to process the full N_b symbols transmitted at one instant through all antennas, rather than the $N_b a$ bits sent per each antenna. The equivalence of transmitted symbols and t_i can be obtained by processing **Hs** for all possible **s**.

Finally, the term $p(\hat{r}/r = t_i)$ is the probability that we have received the symbol r, knowing that the joint-symbol with fading sent was t_i. This is modeled with the classic gaussian shown in expression 19, which in this system, due to the assumption that the noise is i.i.d. AWGN ($C = \sigma^2 I$), can be simplified as shown in 20 for complex constellations.

$$x : \mathcal{N}(\mu, \mathbf{C}) \Rightarrow f(x) = \frac{1}{\pi|\mathbf{C}|} e^{-(\mathbf{x}-\mu)^H \mathbf{C}^{-1}(\mathbf{x}-\mu)} \tag{19}$$

$$x : \mathcal{N}(\mu, \sigma^2 I) \Rightarrow f(x) = \frac{1}{\pi\sigma^2} e^{-\frac{1}{\sigma^2}\|\mathbf{x}-\mu\|^2} \tag{20}$$

The software implementation that was created for this thesis makes use of a highly optimized version of the MAP demapper. This version, unlike the shown here, uses log likelihood ratios rather than probabilities for the calculation of each step. In any case, the principles are indeed the same than the ones shown in this chapter, the difference being merely the use of equivalent expressions to those used here.

Regarding computational efficiency, the MAP decoder suffers from being a demonstrably NP complete problem. As can be seen in the expresion 17, we need to calculate terms for all N_s symbols, and due to the need to perform joint decoding if capacity want to be achieved, those N_s symbols are 2^{N_b}. This demonstrates that the complexity grows exponentially with the number of bits (N_b) used for each MIMO transmission. Therefore, the complexity of the stage could be expressed as show in 21.

$$\text{MAP Computational Complexity} \propto 2^{N_b} \tag{21}$$

2.3.5 Binary MMSE demapper

Introduction

Due to the impossibility of using MAP demapping for large number of transmitted bits, one of the objectives of this thesis was to find a simple method which could perform this stage in non-exponential time. From the beginning, Minimum Mean Squared Error (MMSE) methods were among the most promising. MMSE demappers have been widely used in a range of applications such as multiuser channels, in which it was necessary to consider the interference of other users, as well as ISI channels, in which the interference was due to the spread in time of the signal itself.

The research carried in this thesis used the work in [12] as a starting point. However, notable differences arise between the typical use of MMSE demapper and the MMSE presented here. The main one is that **instead using MMSE estimates at the symbol level**, which later requires further demapping for constellations larger than BPSK ones, **the MMSE used in this thesis will be able to directly demap at the bit level itself**.

The key to decode arbitrarily large QAM constellations at a bit level using MMSE demapping is the ADD constellation and constellation labeling scheme. Specifically, expression 22 pinpoints the critical usefulness of a method such as the ADD one.

$$\begin{aligned} \mathbf{y} &= \mathbf{Hs} + \mathbf{n} \\ \mathbf{y} &= \mathbf{HAc} + \mathbf{n} \end{aligned} \tag{22}$$

Note that, as **s**=**Ac**, **A** being the alpha matrix of the ADD constellation and labeling scheme and **c** being the vector of transmitted bits, there is a direct, simple algebraic relationship between the received signals **y** and the aforementioned transmitted bits. Extending the expression 22 for the case of a 2×2 MIMO system with $N_b = 4$ results firstly in the equations 23 and in the en in 24.

$$\begin{aligned} \begin{bmatrix} y_0 \\ y_1 \end{bmatrix} &= \begin{bmatrix} h_{00} & h_{01} \\ h_{10} & h_{11} \end{bmatrix} \begin{bmatrix} \alpha_0 & j\alpha_0 & 0 & 0 \\ 0 & 0 & \alpha_0 & j\alpha_0 \end{bmatrix} \begin{bmatrix} c_0 \\ c_1 \\ c_2 \\ c_3 \end{bmatrix} + \begin{bmatrix} n_0 \\ n_1 \end{bmatrix} \\ \begin{bmatrix} y_0 \\ y_1 \end{bmatrix} &= \begin{bmatrix} \alpha_0 h_{00} & j\alpha_0 h_{00} & \alpha_0 h_{01} & \alpha_0 h_{01} \\ \alpha_0 h_{10} & j\alpha_0 h_{10} & \alpha_0 h_{11} & \alpha_0 h_{11} \end{bmatrix} \begin{bmatrix} c_0 \\ c_1 \\ c_2 \\ c_3 \end{bmatrix} + \begin{bmatrix} n_0 \\ n_1 \end{bmatrix} \end{aligned} \tag{23}$$

$$\begin{aligned} y_0 &= \alpha_0 h_{00} c_0 + j\alpha_0 h_{00} c_1 + \alpha_0 h_{01} c_2 + \alpha_0 h_{01} c_3 + n_0 \\ y_1 &= \alpha_0 h_{10} c_0 + j\alpha_0 h_{10} c_1 + \alpha_0 h_{11} c_2 + \alpha_0 h_{11} c_3 + n_1 \end{aligned} \tag{24}$$

From the equation 24 it becomes clear that, thanks to the ADD constellation and labeling scheme, if we want to do a MMSE demap to calculate a bit such as c_1, we can consider that the desired signal is $j\alpha_0 h_{00} c_1$ and $j\alpha_0 h_{10} c_1$, while the rest of the terms of the expression could be interpreted and treated equivalently as:

- Interference due to other transmitting antenna signals, both real and *virtual*. As seen in 2.3.3, ADD constellation and labeling schemes can be interpreted as a $N_{ba}N_t \times N_r$ MIMO system rather than the original $N_t \times N_r$ one.
- Interference due to other *virtual users*. In this case the other *virtual users* would be other equivalent BPSK signals carrying the other bits of the transmission.
- Colored noise.

Definition

In order to find the equations that will have to be used in the binary MMSE demapper, we'll firstly define such equations for the case of MMSE demapping at the symbol level and for a single instant in time t. The starting point will be the MIMO channel equation 25, which, as this demapper evaluates a single time instant can be simplified into 26.

$$\mathbf{y}(t) = \mathbf{H}(t)\mathbf{s}(t) + \mathbf{n}(t) \tag{25}$$

$$\mathbf{y} = \mathbf{H}\mathbf{s} + \mathbf{n} \tag{26}$$

As shown in 27, the idea behind MMSE processing with bias is to find a linear estimate of the symbol transmitted by the k-th antenna, s_k, by multiplying the received signal $\mathbf{y}$ by a vector to be estimated $\mathbf{w}$ plus one bias term which will also be evaluated $\mathbf{b}$. As its own name suggest (Minimum Mean Squared Error), this estimate is then optimized to be the smallest possible in terms of average quadratic error 28. This last expression, taking into account 26 and 27, can be found to be 29.

$$\hat{s}_k = \mathbf{w}_k^H \mathbf{y} + b_k \tag{27}$$

$$MMSE = E_{s_k,\mathbf{n}}\{|s_k - \hat{s}_k|^2\} \tag{28}$$

$$\begin{aligned} MMSE &= E_{s_k,\mathbf{n}}\{|s_k - \hat{s}_k|^2\} \\ &= E\{|s_k - \mathbf{w}_k^H \mathbf{y} - b_k|^2\} \\ &= E\{(s_k - \mathbf{w}_k^H \mathbf{y} - b_k)(s_k - \mathbf{w}_k^H \mathbf{y} - b_k)^H\} \end{aligned} \tag{29}$$

Firstly, in 30 the optimum bias term is find by partial differentiation on the term $\mathbf{b}^*$.

$$\begin{aligned} \frac{\partial MSE}{\partial b_k^*} &= -E\{(s_k - \mathbf{w}_k^H \mathbf{y} - b_k)\} = 0 \\ b_k &= E\{(s_k - \mathbf{w}_k^H \mathbf{y})\} \\ b_k &= E\{s_k\} - \mathbf{w}_k^H E\{\mathbf{y}\} \end{aligned} \tag{30}$$

With the $\mathbf{b}$ term found in 30 it's possible to further simplify 28 as shown in 31.

$$\begin{aligned} MMSE &= E_{s_k,\mathbf{n}}\{|s_k - \hat{s}_k|^2\} \\ &= E\{|s_k - \mathbf{w}_k^H \mathbf{y} - b_k|^2\} \\ &= E\{|s_k - \mathbf{w}_k^H \mathbf{y} - E\{s_k\} + \mathbf{w}_k^H E\{\mathbf{y}\}|^2\} \\ &= E\{|(s_k - E\{s_k\}) - \mathbf{w}_k^H (\mathbf{y} - E\{\mathbf{y}\})|^2\} \\ &= \mathbf{w}_k^H Cov(\mathbf{y},\mathbf{y})\mathbf{w}_k - \mathbf{w}_k^H Cov(\mathbf{y}, s_k) - Cov(s_k,\mathbf{y})\mathbf{w}_k + Cov(s_k, s_k) \end{aligned} \tag{31}$$

Optimizing the equation found in 31 in respect to the linear estimated vector $\mathbf{w}$ it's possible to obtain the expression 32.

$$\begin{aligned} \frac{\partial MSE}{\partial \mathbf{w}_k^*} = 0 \Rightarrow \mathbf{w}_k &= Cov(\mathbf{y},\mathbf{y})^{-1} Cov(\mathbf{y}, s_k) \\ \mathbf{w}_k &= (\sigma_n^2 \mathbf{I} + \mathbf{H} Cov(\mathbf{s},\mathbf{s})\mathbf{H}^H)^{-1} \mathbf{H}\mathbf{d}_k \sigma_s^2 \end{aligned} \tag{32}$$

Where $\mathbf{d}_k$ is the k-th column of the matrix defined by $Cov(\mathbf{s},\mathbf{s})$, k being the transmit antenna number that generated the symbol we want to estimate (i.e: 0 if it was transmitted as the s_0 element of the MIMO equation).

Given that we want the extrinsic information, $E\{s_k\} = 0$ and $\sigma_{s_k}^2 = 1$. These two conditions must also be taken into account when calculating $Cov(\mathbf{s},\mathbf{s})$ and $E\{\mathbf{s}\}$. Finally, both $\mathbf{w}$ and the estimate $\hat{s}_k$ can be rewritten as shown in 33.

$$\begin{aligned} \mathbf{w}_k &= (\sigma_n^2 \mathbf{I} + \mathbf{H} Cov(\mathbf{s},\mathbf{s})\mathbf{H}^H)^{-1} \mathbf{H}\mathbf{d}_k \\ \hat{s}_k &= \mathbf{w}_k^H (\mathbf{y} - \mathbf{H} E\{\mathbf{s}\}) \end{aligned} \tag{33}$$

Once the expression of the estimated symbol, $\hat{s}_k$, has been fully developed, the next step is to consider that the distribution of such estimated symbol is approximately gaussian, $\mathcal{N}(\mu_{s_k}, \sigma_{s_k}^2)$. The expressions 34 and 35 show the mean and variance of such estimated distribution.

$$\begin{aligned}\mu_{s_k} &= \mathbf{w}_k^H(E\{\mathbf{y}/s_k = s_i\} - \mathbf{H}E\{\mathbf{s}\}) \\ \mu_{s_k} &= s_i\mathbf{w}_k^H\mathbf{H}\mathbf{d}_k\end{aligned} \tag{34}$$

$$\begin{aligned}\sigma_{s_k}^2 &= \mathbf{w}_k^H Cov(\mathbf{y}, \mathbf{y}/s_k = s_i)\mathbf{w}_k \\ \sigma_{s_k}^2 &= \mathbf{w}_k^H\mathbf{H}\mathbf{d}_k(1 - \mathbf{d}_k^H\mathbf{H}^H\mathbf{w}_k)\end{aligned} \tag{35}$$

At this point, in order to get the estimated probabilities per bit it'd be needed to perform further demapping of the received symbol $\hat{s}_k$, taking into consideration that the new distribution is the one described above. However, in this thesis the focus is a much simpler approach, with the intention of further simplifying the MMSE so that it performs binary demapping. The key to do this is the equivalence $\mathbf{s} = \mathbf{A}\mathbf{c}$ that holds when the ADD constellation and labeling method is used.

Taking into consideration this equivalence, and redoing each one of the steps described but taking into account that we must directly calculate $\hat{c}_k$, it's easy to obtain the following equations, which, thanks to the deterministic nature of $\mathbf{A}$ are the same than above but with the replacement of $\hat{s}_k$ with $\hat{c}_k$ and $\mathbf{s}$ with $\mathbf{A}\mathbf{c}$. The expressions in 36 fully describe the system.

$$\begin{aligned}E\{c_k\} = 0, \sigma_{c_k}^2 &= 1 \\ \mathbf{w}_k = (\sigma_n^2\mathbf{I} + \mathbf{H}\mathbf{A}Cov(\mathbf{c}, \mathbf{c})\mathbf{A}^H\mathbf{H}^H)^{-1}\mathbf{H}\mathbf{A}\mathbf{d}_k \\ \hat{c}_k &= \mathbf{w}_k^H(\mathbf{y} - \mathbf{H}\mathbf{A}E\{\mathbf{c}\}) \\ \mu_{c_k} &= c_i\mathbf{w}_k^H\mathbf{H}\mathbf{A}\mathbf{d}_k \\ \sigma_{c_k}^2 &= \frac{1}{2}\mathbf{w}_k^H\mathbf{H}\mathbf{A}\mathbf{d}_k(1 - \mathbf{d}_k^H\mathbf{A}^H\mathbf{H}^H\mathbf{w}_k)\end{aligned} \tag{36}$$

Where $\mathbf{d}_k$ this time is the k-th column of $Cov(\mathbf{c}, \mathbf{c})$.

Now $\hat{c}_k$ can be understood as the noisy reception of a binary symbol of a BPSK constellation. The 0.5 multiplier of $\sigma_{c_k}^2$ needs to be added due to the fact that now we need the variance in one dimension (real dimension) rather than both of them. By using log-likelihood ratios, as will be seen in 2.4, a very efficient form of calculation can be devised. The expression 37 shows the development of that expression for this particular case ($c_k \in \{1, -1\}$).

$$\mathrm{LLR}(c_k) = \ln\frac{p(\hat{c}_k/c_k = -1)}{p(\hat{c}_k/c_k = 1)} = \ln\frac{\frac{1}{\sqrt{2\pi}\sigma_{c_k}}e^{-\frac{1}{2\sigma_{c_k}^2}(\mathrm{Re}\{\hat{c}_k\} - \mu_{c_k=1})^2}}{\frac{1}{\sqrt{2\pi}\sigma_{c_k}}e^{-\frac{1}{2\sigma_{c_k}^2}(\mathrm{Re}\{\hat{c}_k\} - \mu_{c_k=0})^2}} = -\frac{2}{\sigma_{c_k}^2}\mu_{c_k}\mathrm{Re}\{\hat{c}_k\} \tag{37}$$

Where Re$\{x\}$ is the real part of x. This is needed as this BPSK system operates entirely on the real plane, and therefore any imaginary information has to be discarded. On the other hand, note that c_k in ADD schemes takes the value of -1 when the bit itself is 1, and 1 when the original bit was 0.

Finally, regarding the complexity of the Binary MMSE demapper itself, firstly it can be seen that the last BPSK demapping stage is trivial, and secondly in the main equations of the MMSE demapping, it is obvious that the size of the matrix $Cov(\mathbf{c}, \mathbf{c})$ grows squarely with N_b. The fact that for each single bit there are operations that grow $N_b \times N_b$ points out to a geometric complexity level, though future implementations could take into consideration that $Cov(\mathbf{c}, \mathbf{c})$ is diagonal and exploit that fact to reduce such complexity. In any case, the expression 38 shows this proportionality for the currently implemented Binary MMSE demapper.

$$\text{MMSE Computational Complexity} \propto N_b * N_b \tag{38}$$

Binary MMSE variant

During the research carried for this thesis, it became apparent that a problem could arise due to the fact that while the original bits fed to the random encoder, **b**, are uncorrelated, the output ones, **c**, are not, as the main purpose of random encoding stage is correlating all levels. This happens because originally we transmitted each c_i element of the**c** vector in the same time instant, and therefore at the MMSE reception we had heavily correlated bits for any given instant. That could jeopardize many operations in the MMSE described in the previous chapter, as they assumed statistical independence.

As will be seen in 2.4, this problem was later seen to be solved by using the intra-block temporal interleaving, which by sending the c_i of each original **c** vector in different time instants, yet within the same error-correcting block timeslot (intra-block), made sure that there was no such correlation, or at least, that the cycle length was much larger.

In any case, for legacy purposes, the variant which was developed is discussed here and is briefly tested in the results section, in which it's shown to be no better than the previous variant plus interleaving.

This variant main feature is the fact that it uses the b bits rather than c bits for the calculation of the statistical operands such as the expectancies and covariances. This is accomplished due to two factors:

1. As will be seen in 2.4, there is a direct and simple $GF(2)$ algebraic relationship between **c** and **b**, specifically **b**=**Dc**, where **D** is the decodification matrix.
2. This $GF(2)$ expression can be converted to a real numbers if we map bits from $\{0, 1\}$ to $\{1, -1\}$, and substitute xor operations with multiplications.

Therefore if, for example, we had a certain $\{0, 1\}$ bit c_4 whose equivalence in b bits is

determined by matrix $\mathbf{D}$ as $c_4 = b_1 \otimes b_3 \otimes b_5$, if we consider these same bits as $\{1, -1\}$ we can calculate this same expression with real field operations as $c_4 = b_1 b_3 b_5$.

This allowed the calculation of the covariances and expectancies of $\mathbf{c}$ in terms of b bits, which are always statistically independent. For example, in the case $c_4 = b_1 b_3 b_5$ and $c_1 = b_1 b_2$, $E\{c_4 c_1\}$ could be correctly calculated as $E\{b_3 b_5 b_2\}$ as $b_1 b_1$ is always 1.

On the one hand, this also removed the need for backward propagation in the random decoding stage, as we operated directly on $\mathbf{b}$ terms for *a priori* calculations, but on the other hand the complexity of the MMSE stage was increased, and, most critically, it made extremely difficult to introduce interleavers between $\mathbf{b}$ and $\mathbf{c}$, something that, as will be seen in 2.4 is needed also for proper random decoding.

2.4 Random Encoding and Decoding Stage

2.4.1 Introduction

This stage is the one in charge to perform the operations needed to ensure that the system reaches the outage capacity of the channel. Lamarca's paper [1] established the fundamental principle required for the operation of this stage. In that paper, it was demonstrated that by mapping each of the codewords to be transmitted by the antennas to a random word, in an invertible way, and finally transmitting it, it was possible to attain the outage capacity of the *quasi-static fading channel*.

This method worked due to the nature of the quasi-static fading channels. In these channels, the fading remains constant for the whole duration of the error-correcting block, affecting the capacity attainable at each MLC level in an uneven way. This means that for a given channel, and a given SNR, a error-correcting rate r_1 would be needed, but for another channel, even at the same SNR, a very different r_2 rate would have to be used. In the end, in order to ensure a specific outage probability (probability that the system is going to be in such a fading that it's not going to operate), this would typically mean that for each MLC we would have to accept the higher rates per level while wasting capacity in the channels in which the needed rate was not so high.

By introducing correlations between all the MLC levels, the random labeling approach created rates that were not as dependent on the present fading, but rather only on the SNR of the system. Figure 4 illustrates this point by comparing the rates needed for a specific MLC level in the case of a system without random labeling stages (orange), and one with the paper's [1] original random labeling stage (blue).

The figure represents an example of the first MLC level (first bit). Each single point represents a different channel; the y axis is the total amount of information rate reached for all levels in the system, while the x axis is the rate achieved for that particular level. Note how, for the random encoder plots, at a certain level of total information rate such as 4.5 bits, the rate needed for the error correcting stage is nearly deterministic, at around 0.125,

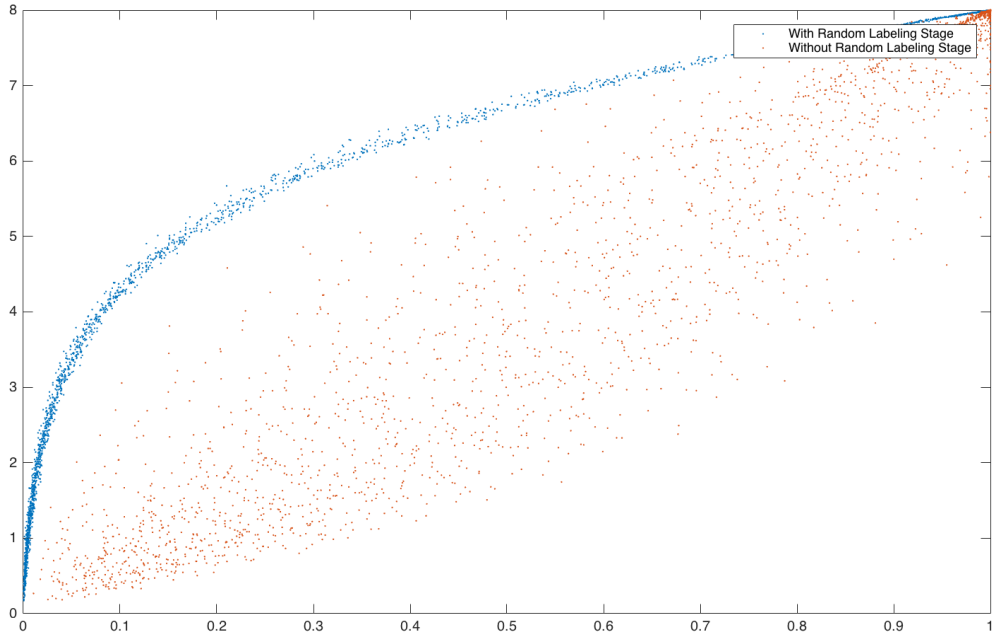

Figure 4: Capacity dispersion graph at first MLC level

as the dispersion of the plots is negligible. On the other hand, in the case of the system without random labeling stage, this dispersion is huge: if we want to be able to decode most of the channels properly, we need to assign a error-correcting rate on the lower band (on the left), despite the fact that there are many channels which would work also with much higher rates. This will become a capacity loss in subsequent MLC levels, as this dispersion plots are mostly symmetric. At the final level of this example (8-th bit), in the case of the random encoding system the rate needed will still be deterministic and much higher rate, while the dispersion of the non-random encoding system will still be wide, and will mean that we will need also a low rate for this level.

Using the naming schemes and message passing steps defined in 2.1, the encoding and decoding functions could be devised as a single function plus its inverse. In the transmitter $\mathbf{b}$ vectors would be firstly mapped into $\mathbf{c}$ vectors, and this mapping would be one-to-one to ensure the existence of its inverse. In the receiver, $\hat{\mathbf{c}}$ vectors would be converted back to $\hat{\mathbf{b}}$, which ideally this would be equal to $\mathbf{b}$, but obviously that would be constrained by the noise and fading present in the channel. The expressions in 39 show this relationship at the highest level.

$$\begin{aligned} \text{Enc}: \quad GF(2)^{1\times N_b} &\rightarrow GF(2)^{1\times N_b} \\ \mathbf{b} &\rightarrow \mathbf{c} = f_{\text{RL}}(t)\,[\mathbf{b}] \end{aligned}$$
$$\begin{aligned} \text{Dec}: \quad GF(2)^{1\times N_b} &\rightarrow GF(2)^{1\times N_b} \\ \hat{\mathbf{c}} &\rightarrow \hat{\mathbf{b}} = f_{\text{RL}}^{-1}(t)\,[\hat{\mathbf{c}}] \end{aligned} \tag{39}$$

Where $f_{\mathrm{RL}}(t)$ would be the encoding function and $f_{\mathrm{RL}}^{-1}(t)$ the decoding one. Note that both functions are dependent on the time interval t as, as we'll see, the encoding and decoding mapping must change several time in a error correcting block in order for the system to operate at it's maximum potential.

The limitation that arose in the aforementioned paper [1] was that the random encoding and decoding stages were implemented in the form of a look-up table, which stored precomputed one-to-one mappings for all 2^{N_b} possible vectors by making random permutations. This approach was efficient in the case N_b was small, but impossible to handle as soon as this parameter increased to not so large levels.

One of the primary objectives of this thesis was to find simple, easy to find $f_{\mathrm{RL}}(t)$ functions, whose invertible were also easy to calculate and which, by themselves, produced no information losses. This step was initially considered to be relatively simple yet, as will be seen in this chapter, was much more complex than initially anticipated.

Since the beginning of the research it was decided that, in order to facilitate the calculations of probabilities, as well as the *soft* message passing between the demapper, random decoder and error-correcting decoding stages, a *factor-graph* implementation using log-likelihood ratios would have to be used. Both llr and factor graph fundamentals used in this thesis are described in section 2.4.2. This was a further constraint to the possible $f_{\mathrm{RL}}(t)$ functions, as they would have to be easy to implement in factor-graph form.

2.4.2 LLR and Factor Graphs

Log-likelihood ratios

Log-likelihood ratios, llr from now onwards, are a representation of probabilities in terms of ratios, which have useful mathematical properties and help to optimize the computational burden needed at certain stages [3]. The thesis' devised system uses llrs instead of probabilities for its calculations.

The conversion between probabilities and llrs is straightforward, and shown in the expressions 40 and 41.

$$
\begin{gathered}
x \in \{0,1\} \subseteq GF(2) \\
\mathrm{P}\left(X=x\right) \in (0,1] \subset \mathbb{R} \\
\mathrm{LLR}\left(X\right) = \ln\left(\frac{\mathrm{P}(X=1)}{\mathrm{P}(X=0)}\right)
\end{gathered}
\tag{40}
$$

$$
\begin{aligned}
\mathrm{P}(x=1) &= \frac{e^{\mathrm{LLR}}(x)}{1+e^{\mathrm{LLR}}(x)} \\
\mathrm{P}(x=0) &= \frac{1}{1+e^{\mathrm{LLR}}(x)}
\end{aligned}
\tag{41}
$$

As can be deduced from these equations, the llr of a variable x will be $+\inf$ if $\mathrm{P}(x=1)=1$, and $-\inf$ if $\mathrm{P}(x=0)=1$, while will be 0 when we have no information of that variable ($\mathrm{P}(x=1)=\mathrm{P}(x=0)=0.5$).

Factor Graphs

Introduction

Factor graphs are efficient mathematical procedures for the computation marginal functions that simplify the summations needed and reuses intermediate values (partial sums).

They are undirected and bipartite graphs that graphically represent the structure and message-passing of the factorization of the stage global function. There are two types of nodes:

1. **Variable Nodes**: Each variable used in the stage has its variable node, which is used to store and extract the final probability of it.
2. **Function Nodes**: They are nodes that connect the different variables and perform a specific function. In this thesis, only two types of function nodes will be shown: and gates and xor gates.

In this thesis, there will be two distinct ways of operation, "forward" operation, in which the messages will go from the top level of the factor graph to the lower level, and "backwards" operation, in which the messages will go from the lowest level to the upper one. The example in 5 and 6 illustrates the equivalent factor graph of a XOR gate, $y = f_{\mathrm{XOR}}(x_1, x_2, x_3)$, where x_i are the input variables and y the output variable, both expressed in terms of llrs as show in the preceding section.

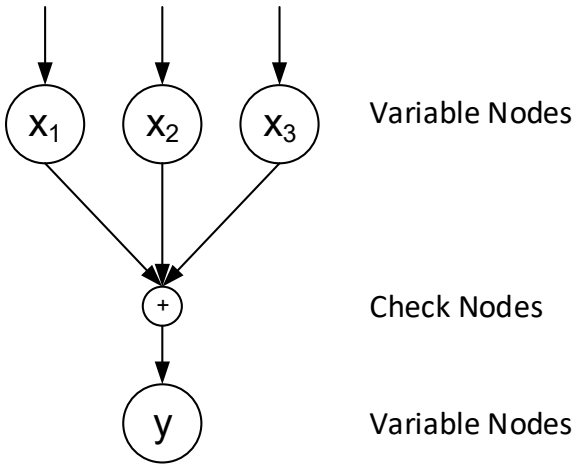

Figure 5: Forward factor graph calculation example

Note how each single variable requires message-passing computation. Therefore, in our specific case, the calculation of the output y requires only one operation, but for the backwards

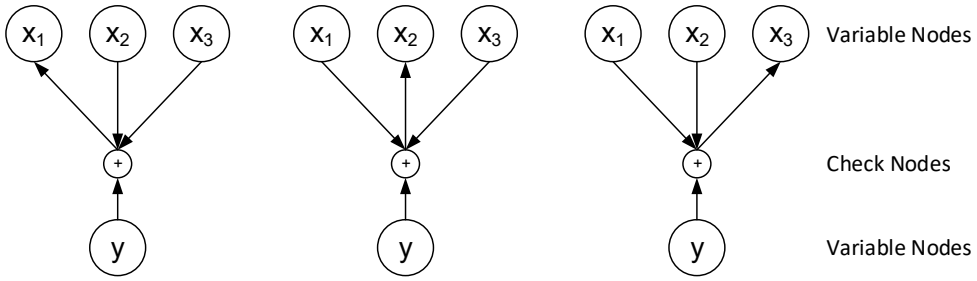

Figure 6: Backward factor graph calculation example

operation we must calculate each x_i independently, three in total. As can be seen in the diagrams, to calculate each output we must process as inputs all of the variables of the nodes except the one to be calculated. In any case, for the computation of a given output of the factor-graph, one should never mix the present stage with the next stage. This means that the calculations for the next stage (next value of the message) has to be done with the previous stage values, without mixing them in the process.

In the next subsection the mathematical expressions of the XOR and AND gates will be shown, and this will be the functions used by the thesis simulation software in order to process the input and output llrs of the stage.

XOR Gate

In order to mathematically define this gate a generic equation describing the inner workings will be expressed. This expression won't be efficient. Secondly, a simple, with few input variables version will be illustrated and optimized for the llr case. Finally, a version allowing large amounts of inputs of that llr optimized gate will be devised.

Let $\hat{y} = f_{\text{xor}}(\hat{x}_1, \hat{x}_2, \ldots, \hat{x}_N)$, with N being the amount of inputs, the function that defines the XOR. $\hat{x}_1, \hat{x}_2, \ldots, \hat{x}_N$ are the llr, *soft* versions of the deterministic boolean variables $x_1, x_2, \ldots, x_N$ used in the encoding process. Let $\mathcal{M}$ be a set of evenly sized vectors $\mathbf{t} = [t_1\ t_2\ \ldots\ t_N] \in GF(2)^{1\times N}$ so that $\sum_{i=1\to N} t_i = 1$, in other words, all N-sized binary words whose bitwise addition is one. On the other hand, let $\mathcal{N}$ be a set of evenly sized vectors $\mathbf{u} = [u_1\ u_2\ \ldots\ u_N] \in GF(2)^{1\times N}$ so that $\sum_{i=1\to N} u_i = 0$, in layman terms, all N-sized binary words whose bitwise addition is zero. With such definitions at hand, the most straightforward method to define the XOR gate as a Log-Likelihood Ratio is to express it as the probability that the input word $(\hat{x}_1, \ldots, \hat{x}_N)$ belongs to one of the words whose addition is one, divided by the probability that the input word belongs to those whose addition is zero:

$$\begin{aligned}\mathrm{LLR}(\hat{y}) &= \ln \frac{\mathrm{P}\left(\sum_{\forall x_i} x_i = 1/(\hat{x}_1, \hat{x}_2, \ldots, \hat{x}_N)\right)}{\mathrm{P}\left(\sum_{\forall x_i} x_i = 0/(\hat{x}_1, \hat{x}_2, \ldots, \hat{x}_N)\right)} \\ &= \frac{\mathrm{P}\left(\bigcup_{\forall \mathbf{t}\in\mathcal{M}} \left(\bigcap_{i=1}^{N}(x_i = t_i)\right)/(\hat{x}_1, \hat{x}_2, \ldots, \hat{x}_N)\right)}{\mathrm{P}\left(\bigcup_{\forall \mathbf{u}\in\mathcal{N}} \left(\bigcap_{i=1}^{N}(x_i = u_i)\right)/(\hat{x}_1, \hat{x}_2, \ldots, \hat{x}_N)\right)}\end{aligned} \tag{42}$$

The expression (42) would allow for optimal, lossless process of any set of variables if properly expanded. However, as the number of input bits of the gate rise the computational burden of such a task quickly becomes insurmountable. This is mostly due to the existence of statistically *dependent* variables. In order to efficiently process this function the input variables have to be statistically independent, and in fact, the encoding/decoding pair matrices are devised to minimize as much as possible such dependencies. From now onwards, all input variables of the XOR gate will be assumed to be independent, hence, the union and intersection of probabilities become:

$$\begin{aligned}\mathrm{P}(A \cap B) &= \mathrm{P}(A)\,\mathrm{P}(B/A) \overset{ind}{=} \mathrm{P}(A)\,\mathrm{P}(B) \\ \mathrm{P}(A \cup B) &= \mathrm{P}(A) + \mathrm{P}(B) - \mathrm{P}(A \cap B) \overset{ind}{=} \mathrm{P}(A) + \mathrm{P}(B)\end{aligned} \tag{43}$$

Using the classic properties of independent variables shown in (43), it becomes possible to express the complex equation in (42) in terms of simple summations and productories instead of complex unions and intersections of sets:

$$\begin{aligned}\mathrm{LLR}(\hat{y}) &= \frac{\mathrm{P}\left(\bigcup_{\forall \mathbf{t}\in\mathcal{M}} \left(\bigcap_{i=1}^{N}(x_i = t_i)\right)/(\hat{x}_1, \hat{x}_2, \ldots, \hat{x}_N)\right)}{\mathrm{P}\left(\bigcup_{\forall \mathbf{u}\in\mathcal{N}} \left(\bigcap_{i=1}^{N}(x_i = u_i)\right)/(\hat{x}_1, \hat{x}_2, \ldots, \hat{x}_N)\right)} \\ &\overset{ind}{=} \frac{\sum_{\forall \mathbf{t}\in\mathcal{M}} \prod_{i=1}^{N} \mathrm{P}(x_i = t_i)}{\sum_{\forall \mathbf{u}\in\mathcal{N}} \prod_{i=1}^{N} \mathrm{P}(x_i = u_i)}\end{aligned} \tag{44}$$

With the equation developed in (44), a two-input gate devised to take llr values both as inputs and as output can be expressed as:

$$
\begin{aligned}
\mathrm{LLR}_{f_x\to\hat{y}} &= \ln\frac{\mathrm{P}(x_1=1)\mathrm{P}(x_2=0)+\mathrm{P}(x_1=0)\mathrm{P}(x_2=1)}{\mathrm{P}(x_1=1)\mathrm{P}(x_2=1)+\mathrm{P}(x_1=0)\mathrm{P}(x_2=0)}\\
&= \ln\frac{\frac{\mathrm{P}(x_1=1)\mathrm{P}(x_2=0)}{\mathrm{P}(x_1=0)\mathrm{P}(x_2=0)}+\frac{\mathrm{P}(x_1=0)\mathrm{P}(x_2=1)}{\mathrm{P}(x_1=0)\mathrm{P}(x_2=0)}}{\frac{\mathrm{P}(x_1=1)\mathrm{P}(x_2=1)}{\mathrm{P}(x_1=0)\mathrm{P}(x_2=0)}+\frac{\mathrm{P}(x_1=0)\mathrm{P}(x_2=0)}{\mathrm{P}(x_1=0)\mathrm{P}(x_2=0)}}\\
&= \ln\frac{\frac{\mathrm{P}(x_1=1)}{\mathrm{P}(x_1=0)}+\frac{\mathrm{P}(x_2=1)}{\mathrm{P}(x_2=0)}}{\frac{\mathrm{P}(x_1=1)\mathrm{P}(x_2=1)}{\mathrm{P}(x_1=0)\mathrm{P}(x_2=0)}+1} = \ln\frac{\mathrm{e}^{\mathrm{LLR}_{x_1\to f_x}}+\mathrm{e}^{\mathrm{LLR}_{x_2\to f_x}}}{1+\mathrm{e}^{\mathrm{LLR}_{x_1\to f_x}+\mathrm{LLR}_{x_2\to f_x}}}\\
&= \ln\frac{\mathrm{e}^{\frac{1}{2}\mathrm{LLR}_{x_1\to f_x}+\frac{1}{2}\mathrm{LLR}_{x_2\to f_x}}}{\mathrm{e}^{\frac{1}{2}\mathrm{LLR}_{x_1\to f_x}+\frac{1}{2}\mathrm{LLR}_{x_2\to f_x}}}\frac{\mathrm{e}^{\frac{1}{2}\mathrm{LLR}_{x_1\to f_x}-\frac{1}{2}\mathrm{LLR}_{x_2\to f_x}}+\mathrm{e}^{-\frac{1}{2}\mathrm{LLR}_{x_1\to f_x}+\frac{1}{2}\mathrm{LLR}_{x_2\to f_x}}}{\mathrm{e}^{-\frac{1}{2}\mathrm{LLR}_{x_1\to f_x}-\frac{1}{2}\mathrm{LLR}_{x_2\to f_x}}+\mathrm{e}^{\frac{1}{2}\mathrm{LLR}_{x_1\to f_x}+\frac{1}{2}\mathrm{LLR}_{x_2\to f_x}}}\\
&= \ln\frac{\cosh\left(\mathrm{LLR}_{x_1\to f_x}-\mathrm{LLR}_{x_2\to f_x}\right)}{\cosh\left(\mathrm{LLR}_{x_1\to f_x}+\mathrm{LLR}_{x_2\to f_x}\right)} = \ln\frac{1-\tanh\left(\frac{1}{2}\mathrm{LLR}_{x_1\to f_x}\right)\tanh\left(\frac{1}{2}\mathrm{LLR}_{x_2\to f_x}\right)}{1+\tanh\left(\frac{1}{2}\mathrm{LLR}_{x_1\to f_x}\right)\tanh\left(\frac{1}{2}\mathrm{LLR}_{x_2\to f_x}\right)}\\
&= -2\mathrm{artanh}\left(\tanh\left(\frac{1}{2}\mathrm{LLR}_{x_1\to f_x}\right)\tanh\left(\frac{1}{2}\mathrm{LLR}_{x_2\to f_x}\right)\right)
\end{aligned}
\tag{45}
$$

If a similar procedure of that shown in (45) is used for increasingly larger number of inputs, the following self contained expression can be determined for a XOR gate of any number of gates:

$$
\mathrm{LLR}_{f_x\to\hat{y}} = (-1)^{N+1}2\mathrm{artanh}\left(\prod_{i=1}^{N}\tanh\left(\frac{1}{2}\mathrm{LLR}_{x_i\to f_x}\right)\right) \tag{46}
$$

Where N is the number of inputs. Note that this expression is valid both in the *forward* and *backward* iteration, for any given inputs and outputs. Obviously, in order to maintain the validity of (46) all inputs should be assumed to be named as $x_1, x_2, \ldots, x_N$ while the output should be considered to be named $\hat{y}$.

AND gate

Let $\hat{y} = f_{\text{and}}(\hat{x}_1, \hat{x}_2, \ldots, \hat{x}_N)$, with N being the amount of inputs, the function that defines the AND. $\hat{x}_1, \hat{x}_2, \ldots, \hat{x}_N$ are the llr, *soft* versions of the deterministic boolean variables $x_1, x_2, \ldots, x_N$ used in the encoding process. Let $\mathcal{M}$ be a set of evenly sized vectors $\mathbf{t} = [t_1\ t_2\ \ldots\ t_N] \in GF(2)^{1\times N}$ so that $\prod_{i=1\to N} t_i \neq 1$, in other words, all N-sized binary words whose bitwise and is not one. Note that this is exactly all possible N-sized binary vectors (2^N) but one, the one whose variables are all set to one. With such definitions at hand, the most straightforward method to define the AND gate as a Log-Likelihood Ratio is to express it as the probability that the input word $(\hat{x}_1, \ldots, \hat{x}_N)$ is the all-ones word, divided by the probability that the input word belongs to those whose bitwise and is zero:

$$\begin{aligned}\mathrm{LLR}(\hat{y}) &= \ln \frac{\mathrm{P}\left(\prod_{\forall x_i} x_i = 1/(\hat{x}_1, \hat{x}_2, \ldots, \hat{x}_N)\right)}{\mathrm{P}\left(\prod_{\forall x_i} x_i = 0/(\hat{x}_1, \hat{x}_2, \ldots, \hat{x}_N)\right)} \\ &= \frac{\mathrm{P}\left(\bigcap_{i=1}^{N}(x_i = 1)/(\hat{x}_1, \hat{x}_2, \ldots, \hat{x}_N)\right)}{\mathrm{P}\left(\bigcup_{\forall \mathbf{t}\in\mathcal{M}}\left(\bigcap_{i=1}^{N}(x_i = t_i)\right)/(\hat{x}_1, \hat{x}_2, \ldots, \hat{x}_N)\right)}\end{aligned} \tag{47}$$

As in the XOR case, the expression (47) would allow for optimal, lossless process of any set of variables if properly expanded. Once again, it's not computationally viable unless we assume independence of variables.

Using such properties, it becomes possible to express the complex equation in (47) in terms of simple summations and productories instead of complex unions and intersections of sets:

$$\begin{aligned}\mathrm{LLR}(\hat{y}) &= \frac{\mathrm{P}\left(\bigcap_{i=1}^{N}(x_i = 1)/(\hat{x}_1, \hat{x}_2, \ldots, \hat{x}_N)\right)}{\mathrm{P}\left(\bigcup_{\forall \mathbf{t}\in\mathcal{M}}\left(\bigcap_{i=1}^{N}(x_i = t_i)\right)/(\hat{x}_1, \hat{x}_2, \ldots, \hat{x}_N)\right)} \\ &\overset{ind}{=} \frac{\prod_{i=1}^{N}\mathrm{P}\left(x_i = 1\right)}{\sum_{\forall \mathbf{t}\in\mathcal{M}}\prod_{i=1}^{N}\mathrm{P}\left(x_i = t_i\right)}\end{aligned} \tag{48}$$

With the equation developed in (48), a two-input AND gate devised to take llr values both as inputs and as output can be expressed as:

$$\begin{aligned}\mathrm{LLR}_{f_a\to\hat{y}} &= \ln \frac{\mathrm{P}(x_1 = 1)\mathrm{P}(x_2 = 1)}{\mathrm{P}(x_1 = 0)\mathrm{P}(x_2 = 1) + \mathrm{P}(x_1 = 1)\mathrm{P}(x_2 = 0) + \mathrm{P}(x_1 = 0)\mathrm{P}(x_2 = 0)} \\ &= \ln \frac{\frac{\mathrm{P}(x_1=1)\mathrm{P}(x_2=1)}{\mathrm{P}(x_1=0)\mathrm{P}(x_2=0)}}{\frac{\mathrm{P}(x_1=0)\mathrm{P}(x_2=1)+\mathrm{P}(x_1=1)\mathrm{P}(x_2=0)+\mathrm{P}(x_1=0)\mathrm{P}(x_2=0)}{\mathrm{P}(x_1=0)\mathrm{P}(x_2=0)}} \\ &= \ln \frac{\frac{\mathrm{P}(x_1=1)}{\mathrm{P}(x_1=0)}\frac{\mathrm{P}(x_2=1)}{\mathrm{P}(x_2=0)}}{\frac{\mathrm{P}(x_2=1)}{\mathrm{P}(x_2=0)} + \frac{\mathrm{P}(x_1=1)}{\mathrm{P}(x_1=0)} + 1} = \ln \frac{\mathrm{e}^{\mathrm{LLR}_{x_1\to f_a}+\mathrm{LLR}_{x_2\to f_a}}}{\mathrm{e}^{\mathrm{LLR}_{x_1\to f_a}} + \mathrm{e}^{\mathrm{LLR}_{x_2\to f_a}} + 1} \\ &= \mathrm{LLR}_{x_1\to f_a} + \mathrm{LLR}_{x_2\to f_a} - \ln\left(1 + \mathrm{LLR}_{x_1\to f_a} + \mathrm{LLR}_{x_2\to f_a}\right)\end{aligned} \tag{49}$$

Unlike the XOR case, this gate is asymmetric. The equations for the *backwards* calculations won't be the same than in the *forward* case. In order to define them, let $\hat{x}_1 = f_{\text{and}}(\hat{y}, \hat{x}_2, \ldots, \hat{x}_N)$, with N being the amount of inputs, the function that defines the backwards AND operation. $\hat{y}, \hat{x}_2, \ldots, \hat{x}_N$ are the llr, *soft* versions of the deterministic boolean variables $y, x_2, \ldots, x_N$ used in the encoding process. Let $\mathcal{M}$ be a set of evenly sized vectors $\mathbf{t} = [t_1\ t_2\ \ldots\ t_N] \in GF(2)^{1\times N}$ that contains all the $[\hat{y}\ \hat{x}_2\ \ldots\ \hat{x}_N]$ vectors for which can possibly be 1. On the other hand, let $\mathcal{N}$ be a set of evenly sized vectors $\mathbf{u} = [u_1\ u_2\ \ldots\ u_N] \in GF(2)^{1\times N}$ containing all the $[\hat{y}\ \hat{x}_2\ \ldots\ \hat{x}_N]$ vectors for which a could be zero. With such definitions at hand, the most straightforward method to define the backwards AND gate as a Log-Likelihood Ratio is to express it as the probability that the input word $[\hat{y}, \hat{x}_2, \ldots, \hat{x}_N]$ belongs to one of the words whose addition is one, divided by the probability that the input word belongs to those whose addition is zero:

$$\text{LLR}(\hat{x}_j) = \frac{\text{P}\left(\bigcup_{\forall \mathbf{t}\in\mathcal{M}} \left((y = t_j) \bigcap_{i=1, i\neq j}^{N} (x_i = t_i)\right) / (\hat{x}_1, \hat{x}_2, \ldots, \hat{x}_N)\right)}{\text{P}\left(\bigcup_{\forall \mathbf{u}\in\mathcal{N}} \left((y = u_j) \bigcap_{i=1, i\neq j}^{N} (x_i = u_i)\right) / (\hat{x}_1, \hat{x}_2, \ldots, \hat{x}_N)\right)} \tag{50}$$

Where j defines the position in the vectors $\mathbf{u}$ and $\mathbf{t}$ in which we have placed the value of y. In the example we've defined in the previous paragraph, $j = 1$, as we intend to calculate x_1. Once again, assuming statistical independence, and using the classic properties of independent variables shown in (43), it becomes possible to express the complex equation in (50) in terms of simple summations and productories instead of complex unions and intersections of sets:

$$\begin{aligned}\text{LLR}(\hat{x}_j) &= \frac{\text{P}\left(\bigcup_{\forall \mathbf{t}\in\mathcal{M}} \left((y = t_j) \bigcap_{i=1, i\neq j}^{N} (x_i = t_i)\right) / (\hat{x}_1, \hat{x}_2, \ldots, \hat{x}_N)\right)}{\text{P}\left(\bigcup_{\forall \mathbf{u}\in\mathcal{N}} \left((y = u_j) \bigcap_{i=1, i\neq j}^{N} (x_i = u_i)\right) / (\hat{x}_1, \hat{x}_2, \ldots, \hat{x}_N)\right)} \\ &\overset{ind}{=} \frac{\sum_{\forall \mathbf{t}\in\mathcal{M}} \text{P}(y = t_j) \prod_{i=1, i\neq j}^{N} \text{P}(x_i = t_i)}{\sum_{\forall \mathbf{u}\in\mathcal{N}} \text{P}(y = u_j) \prod_{i=1, i\neq j}^{N} \text{P}(x_i = u_i)}\end{aligned} \tag{51}$$

With the equation developed in (51), a two-input gate devised to take llr values both as inputs and as output can be expressed as:

$$\begin{aligned}\mathrm{LLR}_{f_a\to x_1} &= \ln\frac{\mathrm{P}\left(x_2=0\right)\mathrm{P}\left(y=0\right)+\mathrm{P}\left(x_2=1\right)\mathrm{P}\left(y=1\right)}{\mathrm{P}\left(x_2=0\right)\mathrm{P}\left(y=0\right)+\mathrm{P}\left(x_2=1\right)\mathrm{P}\left(y=0\right)} = \ln\frac{1+\frac{\mathrm{P}(x_2=1)\mathrm{P}(x_2=1)}{\mathrm{P}(x_2=0)\mathrm{P}(y=0)}}{1+\frac{\mathrm{P}(x_2=1)\mathrm{P}(y=0)}{\mathrm{P}(x_2=0)\mathrm{P}(y=0)}} \\ &= \ln\frac{1+\mathrm{e}^{\mathrm{LLR}_{f_a\to x_2}+\mathrm{LLR}_{f_a\to y}}}{1+\mathrm{e}^{\mathrm{LLR}_{f_a\to x_2}}}\end{aligned} \tag{52}$$

And conversely, for x_2:

$$\mathrm{LLR}_{f_a\to x_2} = \ln\frac{1+\mathrm{e}^{\mathrm{LLR}_{f_a\to x_1}+\mathrm{LLR}_{f_a\to y}}}{1+\mathrm{e}^{\mathrm{LLR}_{f_a\to x_1}}} \tag{53}$$

Factor-graph Cycles

Factor-graph cycles are loops in the structure of the factor graph [3]. These loops are undesirable as they always produce information losses in the message-passing algorithm, due to the fact that they represent dependence between the factor-graph variables. As can be seen in the preceding subsection, all factor graph function node calculations are done taking into account that there is complete statistical independence of the variables used in the node.

In this thesis, we'll face two types of cycles, which in this thesis we will name as:

1. **Intra-Stage Cycles**: Cycles due to the factor-graph structure of the stage itself.
2. **Inter-Stage Cycles**: Cycles due to the fact that, the input variables fed to the stage are not statistically independent to begin with.

In order to categorize the cycles, it's typical to use the so-called cycle-length. This value indicates the number of forward and backward iterations needed for the cycle itself to appear. In general, the smaller the cycle-length, the greatest its impact.

The example in 7 shows a length 2 cycle within the structure of a possible stage of our random decoder. Note that at the second iteration, the values of c_1 and c_2 will not be anymore statistically independent.

2.4.3 Researched but discarded methods

Introduction

This section's main purpose is to summarize, in a brief way, the most important methods that were at one point evaluated as the solutions for this project, but that finally were rejected. A description of the method is offered, as well as the reasons of why it was not deemed the best solution for the thesis.

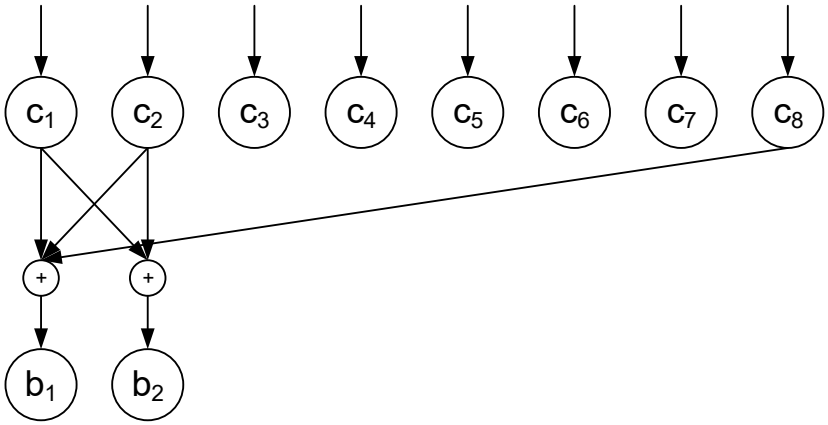

Figure 7: Length 2 cycle between c1 and c2

The methods are ordered chronologically, with the first presented one being the oldest, first evaluated one and the last one the last that was evaluated with the exception of the final proposed method, which is described in the next section 2.4.4.

Standard Random Generators

In the first step of the thesis research, we looked for well know, standard pseudo-random generators. These included linear congruential generators, non-linear congruential generators, linear feedback shift register (LFSR) and the famous Mersenne Twister.

The random sequence produced by these generators was, as would be expected, very good, with the results in terms of outage capacity being on the same level than the last one, Mersenne Twister, which is the one used by MATLAB and therefore the one in which all 'pure random' code testing is based on.

These generators had, on the other hand, either one or two problems that made them extremely difficult to implement in a factor-graph without inherent losses:

1. They needed numerous iterations for each randomly generated number, such as the case of the lineal congruential generators and LFSRs, which would mean several stacked factor graphs which would involve numerous cycles and thereby information losses.
2. They used difficult non-linear operations, such as in the non-linear congruential generator, which used exponentiations. These functions can't be easily be added to a factor graph.

Due to either of these two reasons, each single one of the analyzed generators had to be discarded.

Original Lineal Codes

From the early beginning, one of the most important candidates as a solution for this problem

were standard lineal codes. These codes would simply map a N_b sized set of original bits $\mathbf{b}$ to a N_b sized set of randomly coded bits $\mathbf{c}$ by using a $N_b \times N_b$ matrix $\mathbf{G}$. $\mathbf{G}$ would have to be a full-rank matrix so that the code itself was invertible. On the receiver side, the received $\hat{\mathbf{c}}$ bits would be converted back to the $\hat{\mathbf{b}}$ by using the $\mathbf{D}$ matrix, the inverse of matrix $\mathbf{G}$, in a factor-graph scheme. Expressions in 54 show the relationships between the binary vectors.

$$\begin{aligned} \mathbf{c} &= \mathbf{G}\mathbf{b} \\ \hat{\mathbf{b}} &= \mathbf{D}\hat{\mathbf{c}} \end{aligned} \tag{54}$$

For each error-correcting stage block, the $\mathbf{G}$ and $\mathbf{D}$ matrices changes several times, in order to make the code as random as possible.

At this stage of research, this system had several limitations which, by the time, seemed to point out that something more complex was needed. The issues that this system showed up where the following:

1. $\mathbf{G}$ matrices had to be searched by brute force, from random attempts, at this stage. This was not a problem for small number of bits (small N_b), but the problem increased exponentially with the number of them. This also limited the number of lineal codes available.

2. The performance of the stage in terms of outage capacity was, at this point, sub par. Up to 10 different lineal codes were tested per each error-correcting block, but the performance in dispersion and outage capacity was far from the 10 pure random codes tests (see the Results 1 section for more information).

3. The fact that the $\mathbf{G}$ matrices where generated randomly until a full-rank one was found meant that no attention to cycles were paid. The matrices generated were very prone to intra-stage cycles, which in turn meant information losses due to the stage itself.

For these reasons, and specially for the second one, this method was discarded at this time.

Selected Lineal Codes

In order to improve the previous method flaw number 2, some tests were carried out to try to improve the *randomness* of the resulting codes.

Using MATLAB's 'runtest' functions, which allow to test the apparent randomness of one vector, the best linear codes found were manually chosen and then simulated against the 'pure random' codes.

The results were disappointing: almost no difference was seen in comparison to the randomly generated codes. At this point in the research it was believed that linear random codes alone would not suffice for this problem.

Original Non-Lineal Codes

In order to introduce more *randomness* in the sequence, a new system was devised. This system would not only use linear operations, but also second-order non-linear ones. These codes would once again map a N_b sized set of original bits $\mathbf{b}$ to a N_b sized set of randomly coded bits $\mathbf{c}$ by using a $N_b \times (N_b * N_b)$ matrix $\mathbf{G}$. On the receiver side, the received $\hat{\mathbf{c}}$ bits would be converted back to the $\hat{\mathbf{b}}$ by using the $\mathbf{D}$ matrix, a pseudo-inverse of matrix $\mathbf{G}$, in a factor-graph scheme. Expressions in 55 show the relationships between the binary vectors.

$$\begin{aligned} \mathbf{c} &= \mathbf{G}(\mathbf{b} \otimes \mathbf{b}) \\ \hat{\mathbf{b}} &= \mathbf{D}(\hat{\mathbf{c}} \otimes \hat{\mathbf{c}}) \end{aligned} \tag{55}$$

Where $\otimes$ is the Kronecker tensor product; in this system we where calculating all first and second order non-lineal $\mathbf{b}$ and $\mathbf{c}$ elements.

The results with this system where mixed:

- In term of outage capacity, the simulations showed that the non-lineal codes where indeed superior to the lineal ones, yet still well under the performance of the 'pure random' codes generated by the MATLAB Mersenne Twister.
- $\mathbf{G}$ matrices had to be search by brute force, and if the resulting code was invertible, then the needed $\mathbf{D}$ matrix had also to be searched by brute force. This meant that the system was not at this point scalable to large N_bs.
- The above fact also made quite difficult to keep intra-stage cycles at bay.

Stacked Non-Lineal Codes

In order to improve the outage capacity of the non-linear codes, several non-linear stages where staged one after the other. The inverse could easily be calculated by using the inverse stages in the reverse order. Figure 8 shows the stage order graphically.

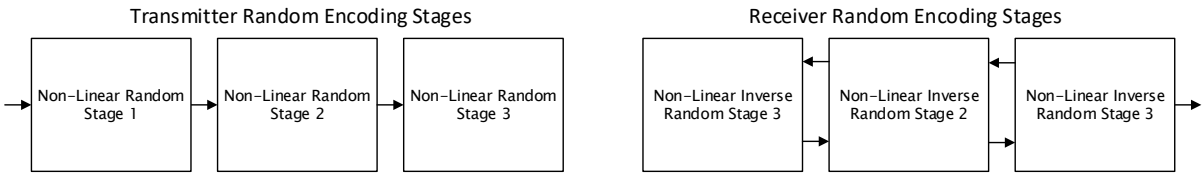

Figure 8: Transmitter and Receiver Stacked Non-Linear Codes Schematic

This system produced higher-order non-linearities. It's worth nothing that, if used with lineal codes, it would be useless, as the result of several stacked lineal codes would still be an equivalent lineal code.

The results were once again mixed:

- In terms of outage capacity, the codes generated by this system were as good as the pure random ones. The higher order non-linealities produced high quality randomness.

- Unfortunately, the stacked stages would mean that a lot of intra-block cycles would be introduced. This led to information losses when this stage was processed as a soft stage rather than a lock-up table.

Pseudotriangular and Recursive Non-Lineal Codes with Interleaving

As the research progressed, two new important discoveries would be made:

- By forcing **G** to have a special structure, it was possible to quickly and reliably create invertible **G**, whose pseudoinverse **D** was easy to obtain, and furthermore it would be easy to keep cycles at bay.

- If instead of sending the coded word **c** directly, we introduced a spatial interleaving between the N_b bits sent in a given instant, the performance in terms of outage capacity would increase dramatically.

This new system was devised so that both **b** and already processed **c** bits would be used in further **c** bit calculations.

At one point in time, this system was considered the solution for the problem presented and therefore it was properly documented. In this thesis Annex 7 a full description of the system can be seen. It's not summarized in this chapter as its definition is rather complex.

Pseudotriangular and Recursive Lineal-Codes With Interleaving

After the discoveries found in the previous step, this very same system was also tested for linear codes. The linear codes also greatly benefited from an spatial interleaver, and the scheme also provided very simple ways to create **G** matrices and they inverse matrices **D**, while at the same time keeping in check the intra-block cycles.

In a very similar, yet more simple way than the previous method. Once again, $\mathbf{b} = [b_1\ b_2\ b_3\ \dots\ b_{N_b}]^T$ is the N_b input bits of the coder, while the vector $\mathbf{c} = [c_1\ c_2\ c_3\ \dots\ c_{N_b}]^T$ represents the coded output bits of the stage. We define the compound vector $[\mathbf{c}\ \mathbf{b}]^T = [c_1\ c_2\ c_3\ \dots\ c_{N_b}\ b_1\ b_2\ b_3\ \dots\ b_{N_b}]^T$ and the $N_b \times (2.N_b)$ matrix **G** such that:

$$\mathbf{G} = \begin{bmatrix} 0 & \cdots & 0 & 0 & g_{11} & \cdots & g_{1(N_b-2)} & g_{1(N_b-1)} & 1 \\ 0 & \cdots & 0 & g_{21} & g_{22} & \cdots & g_{2(N_b-1)} & 1 & 0 \\ 0 & \cdots & g_{31} & g_{32} & g_{33} & \cdots & 1 & 0 & 0 \\ \vdots & \ddots & \vdots & \vdots & \vdots & \ddots & \vdots & \vdots & \vdots \\ 0 & \cdots & g_{N_b(N_b-2)} & g_{N_b(N_b-1)} & 1 & \cdots & 0 & 0 & 0 \end{bmatrix} \tag{56}$$

Where **G** defines which input and output bits will be used for each output bit. Given these vectors and matrix the system's output can be calculated from the following expressions:

$$\begin{bmatrix} 0 & \cdots & 0 & 1 \\ 0 & \cdots & 1 & 0 \\ \vdots & \ddots & \vdots & \vdots \\ 1 & \cdots & 0 & 0 \end{bmatrix} \mathbf{c} = \mathbf{G} \begin{bmatrix} \mathbf{c} \\ \mathbf{b} \end{bmatrix} \tag{57}$$

$$\begin{bmatrix} c_{N_b} \\ c_{N_b-1} \\ c_{N_b-2} \\ \vdots \\ c_1 \end{bmatrix} = \begin{cases} b_1 g_{11} & + & b_2 g_{12} & + & \cdots & + & b_{N_b-1} g_{1(N_b-1)} & + & b_{N_b} \\ c_{N_b} g_{21} & + & b_1 g_{22} & + & \cdots & + & b_{N_b-2} g_{2(N_b-1)} & + & b_{N_b-1} \\ c_{N_b-1} g_{31} & + & c_{N_b} g_{32} & + & \cdots & + & b_{N_b-3} g_{3(N_b-1)} & + & b_{N_b-2} \\ \vdots & & \vdots & & \ddots & & \vdots & & \vdots \\ c_2 g_{N_b 1} & + & c_3 g_{N_b 2} & + & \cdots & + & c_{N_b-1} g_{N_b(N_b-1)} & + & b_1 \end{cases} \tag{58}$$

Note how c_{N_b} is an independent term which depends entirely on known matrix **G** and vector **b**. Each element $c_i, i < N_b$ depends, at most, on the results of already calculated elements $[c_{i+1}\ c_{i+2}\ \cdots\ c_{N_b}]$ besides **G** and **b**. This allows a straightforward, iterative codification of each output. Furthermore, as can be seen in the last expressions, the inverse is very easy to find using this structure.

Once again, as in the previous method, this system was thought to be the solution to be presented in the thesis at one point in time. As such, it was properly documented, and can be seen in the thesis Annex 7, in which much more detailed information is presented and the decoder structure shown.

2.4.4 Final proposed methods for the stage

Introduction

After all of the research carried in 2.4.3, a last attempt at improvement was carried. This attempt involved introducing the new discoveries made during the research of the Linear and Non-Linear Pseudotriangular and Recursive Codes with Interleaving into the original lineal code structures. These discoveries were the following ones:

- The introduction of the spatial interleaving of the coded **c** bits, whose N_b bits would be shuffled randomly at each time instant before the transmission. This greatly improved the outage capacity of the system.
- The introduction of a specific structure for the **G** matrix that ensured easy construction of codes as well as their inverses, along the possibility of reducing cycles quickly.

The resulting method would follow the expressions 59 shown in the original lineal code subsection of 2.4.3. Namely, these codes would simply map a N_b sized set of original bits **b**

to a N_b sized set of randomly coded bits $\mathbf{c}$ by using a $N_b \times N_b$ matrix $\mathbf{G}$. On the receiver side, the received $\hat{\mathbf{c}}$ bits would be converted back to the $\hat{\mathbf{b}}$ by using the $\mathbf{D}$ matrix, the inverse of matrix $\mathbf{G}$, in a factor-graph scheme.

$$\begin{aligned} \mathbf{c} &= \mathbf{G}\mathbf{b} \\ \hat{\mathbf{b}} &= \mathbf{D}\hat{\mathbf{c}} \end{aligned} \tag{59}$$

As mentioned above, there would be, though, two key differences between the original lineal codes studied and the final proposed ones. First of all the $\mathbf{G}$ matrix and its corresponding $\mathbf{D}$ matrix would strictly be **triangular matrices**. Both of them diagonal bits would always be 1, thereby ensuring that the matrices where full-rank matrices and hence invertible.

On the other hand, the coded bits $\mathbf{c}$ would not be directly transmitted, but rather interleaved and then transmitted. This interleaving would be undone at the receiver, just before converting the $\mathbf{c}$ vectors into the original $\mathbf{b}$.

Both $\mathbf{G}$ and $\mathbf{D}$ matrices, as well as interleavers following the explained principles are detailed in the next subsequent sections.

Bit Interleavers

Introduction

As mentioned in 2.4.3, spatial interleavers introduced after the random coding stage, and before the random decoding stage, greatly improved the outage capacity of the overall system.

The reason behind this is that without this stage, both lineal and non-lineal codes typically depended on a few coded bits ($\mathbf{c}$) in order to process each original bit b_i. Without spatial interleaving before the transmission, this meant that the aforementioned coded bits could all be affected by the channel's fading in a similar fashion, and this detrimental effect would be maintained during the whole duration of the error-correcting block. On a different channel, though, these bits could be unaffected by fading, and therefore the behavior of the system in terms of needed rates would be very different; this is exactly what we intended to avoid.

By introducing the spatial interleaver, the $\mathbf{c}$ bits which would be needed to decode each b_i change at every single channel transmission regardless of the $\mathbf{D}$ matrix in use. Therefore, even if some $\mathbf{c}$ bits are in deep fading, the b_i will never depend on them for the whole duration of the error-correcting block, and the information losses due to the fading will be **averaged** between all of the N_b received bits.

However, this section will not only introduce spatial interleavers, but also intra-block *temporal* ones. These intra-block temporal interleavers have another, entirely different purpose: to avoid what we called inter-stage cycles. This is, the fact that we need statistically independent c bits as input of our random decoding stages unless we want to incur in information losses. This point is specially important for the MMSE demapper (2.3.5), and in fact a special MMSE was originally designed in order to counteract this. The simple, efficient use

of intra-block temporal interleavers removed the need for the second MMSE variant, much more complex in nature.

Note that this **intra-block temporal interleaving** is **interleaving done between different instants of the same error-correcting block**. This is a stark contrast with the temporal interleaving used in other schemes such as D-BLAST or TST, as this method introduces no further delays to the error-correcting decoding stage processing.

All of the interleavers tested in simulation will be shown in this section. In order to do so, an example is set in the form of a system with $N_b = 4$, and with a total of six timeslots for each error-correcting block. Without any interleaving, this hypothetical system could be represented as follows in terms of transmitted bits:

$$\begin{array}{cccccc} c_0^0 & c_0^1 & c_0^2 & c_0^3 & c_0^4 & c_0^5 \\ c_1^0 & c_1^1 & c_1^2 & c_1^3 & c_1^4 & c_1^5 \\ c_2^0 & c_2^1 & c_2^2 & c_2^3 & c_2^4 & c_2^5 \\ c_3^0 & c_3^1 & c_3^2 & c_3^3 & c_3^4 & c_3^5 \\ t=1 & t=2 & t=3 & t=4 & t=5 & t=6 \end{array} \tag{60}$$

Each column represents a $\mathbf{c}$ vector to be transmitted at a given time instant t; the left side would be the first time instant to be transmitted ($t = 1$) and the right side the last one ($t = 6$). c_i^t would be the i coded bit that originally was going to be transmitted at time interval t.

The expression shown above would be, obviously, the one corresponding to a system without any kind of interleaving.

Spatial Interleaving

Spatial interleaving is done by random randomly shuffling the elements of each column of 60. Therefore, the i element placement changes, but the original time interval remains the same. The following representation would describe an example of this spatial interleaving:

$$\begin{array}{cccccc} c_3^0 & c_1^1 & c_0^2 & c_2^3 & c_3^4 & c_0^5 \\ c_1^0 & c_3^1 & c_3^2 & c_1^3 & c_2^4 & c_3^5 \\ c_0^0 & c_2^1 & c_1^2 & c_0^3 & c_1^4 & c_2^5 \\ c_2^0 & c_0^1 & c_2^2 & c_3^3 & c_0^4 & c_1^5 \\ t=1 & t=2 & t=3 & t=4 & t=5 & t=6 \end{array} \tag{61}$$

Intra-block Temporal Interleaving

Intra-block Temporal interleaving is accomplished by randomly shuffling the elements of each row of 60. Therefore i element placing remains constant, but the original time interval is not the final time interval in which the transmission is done. The following expressions show an

example of such interleaving:

$$\begin{array}{cccccc} c_0^4 & c_0^0 & c_0^1 & c_0^3 & c_0^2 & c_0^5 \\ c_1^1 & c_1^5 & c_1^4 & c_1^3 & c_1^4 & c_1^2 \\ c_2^5 & c_2^1 & c_2^4 & c_2^3 & c_2^2 & c_2^0 \\ c_3^0 & c_3^3 & c_3^4 & c_3^1 & c_3^2 & c_3^5 \\ t=1 & t=2 & t=3 & t=4 & t=5 & t=6 \end{array} \tag{62}$$

Spatial Intra-block Temporal Interleaving

Spatial Intra-block Temporal interleaving is done by randomly shuffling the elements both in rows and columns of 60. Neither i element placing remains constant, nor the original time interval is the final time interval in which the transmission is done.

Spatial Intra-block Temporal Smart Interleaving

Spatial Intra-block Temporal Smart Interleaving is a method devised for this system, in which firstly a Temporal Interleaving is done, and then a Spatial Interleaving is carried out. The main difference with the Spatial Intra-block Temporal Interleaving is that while the second one would allow for several i-th bits to be transmitted at the same time interval, this one wouldn't. This has some effect on performance, as will be seen in the Results chapter.

G and D matrices selection

Introduction

This thesis will simulate two different types of **G** and **D** matrices:

1. Upper Triangular **G** and **D** matrices

2. Lower Triangular **G** and **D** matrices

Both types of matrices will have diagonal elements equal to one, i.e. $g_{ij} = 1 \forall i = j$ and $d_{ij} = 1 \forall i = j$. This, in the case of triangular matrices, ensures that the matrices have full-rank and therefore that they are invertible.

Upper Triangular Matrices

These matrices will follow the scheme shown in the following expressions:

$$\mathbf{G} = \begin{bmatrix} 1 & g_{01} & g_{02} & \dots & g_{0N_b} \\ 0 & 1 & g_{12} & \dots & g_{1N_b} \\ 0 & 0 & 1 & \dots & g_{2N_b} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \dots & 1 \end{bmatrix}$$
$$\mathbf{D} = \begin{bmatrix} 1 & d_{01} & d_{02} & \dots & d_{0N_b} \\ 0 & 1 & d_{12} & \dots & d_{1N_b} \\ 0 & 0 & 1 & \dots & d_{2N_b} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \dots & 1 \end{bmatrix} \tag{63}$$

Typically, the d_{ij} values will be first calculated and then the inverse of the $\mathbf{D}$ matrix performed in order to find the matrix $\mathbf{G}$. Simulations have proved that moderately sparse random $\mathbf{D}$ matrices, with careful selection to avoid small length cycles, provide reasonable performances.

Lower Triangular Matrices

These matrices will follow the scheme shown in the following expressions:

$$\mathbf{G} = \begin{bmatrix} 1 & 0 & 0 & \dots & 0 \\ g_{10} & 1 & 0 & \dots & 0 \\ g_{20} & g_{21} & 1 & \dots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ g_{N_b0} & g_{N_b1} & g_{N_b2} & \dots & 1 \end{bmatrix}$$
$$\mathbf{D} = \begin{bmatrix} 1 & 0 & 0 & \dots & 0 \\ d_{10} & 1 & 0 & \dots & 0 \\ d_{20} & d_{21} & 1 & \dots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ d_{N_b0} & d_{N_b1} & d_{N_b2} & \dots & 1 \end{bmatrix} \tag{64}$$

Typically, the d_{ij} values will be first calculated and then the inverse of the $\mathbf{D}$ matrix performed in order to find the matrix $\mathbf{G}$. Simulations have proved that very sparse random $\mathbf{D}$ matrices, with careful selection to avoid small length cycles, provide good performances.

Decoding Architectures

Introduction

This section will focus on the different architectures that we'll test in order to decode each of the original b_i bits. In the standard MLC architecture, this is not open for interpretation: at

each level, only one bit is demapped and subsequently sent to the error-correcting decoding stages.

However, in the case of the random coding, a peculiar situation situation arises. In order to decode each b_i bit, in most situation we'll need several c_j bits from the demapping stage. This gives a range of possible interpretations on how to proceed with the demapping and decoding at each stage.

In this thesis, we've tested four different methods to perform this:

1. For each new calculated b_i bit, demap again all c_j bits needed for this b_i bit and for the previously calculated ones.

2. For each new calculated b_i bit, demap again all c_j bits needed for this b_i bit only.

3. For each new calculated b_i bit, demap for the first time only the c_j bits not demapped previously.

4. Same conditions than in previous item, but once all b_is are calculated, using the a priori information obtained, start again the MLC decoding proces at b_0 (iterating a defined number of times).

Each of these architectures are explained with figures in the subsequent sections, and their performance evaluated and analyzed in the Results chapter.

Architecture 1: Full demapping of all needed bits

This random decoder architecture is the most complex one:

1. At the beginning of the process of each MLC level (b_i calculation using the received vector $\mathbf{c}$), firstly we clear all of the previous values to 0 inside the factor graph, with the exception of the calculated b_i and c_j variable nodes.

2. With the *apriori* information that was generated while decoding previous MLC levels, demap all of the c_j bits needed to decode b_i, as well as any previous b_i bit (i-1, i-2, etc); that is, we demap once again everything that was needed to demap to reach the present MLC level.

3. Use the factor graph as show in 9: Firstly calculate again b_0, but **with the error-correcting stage llr fixed**, do a backwards propagation, and continue with b_1. Note that for the b_i level we only calculate the links attached to the b_i variable node: the rest are considered fixed in previous steps. We do forward and backward propagations until we reach b_i.

4. Calculate b_i with all the generated information.

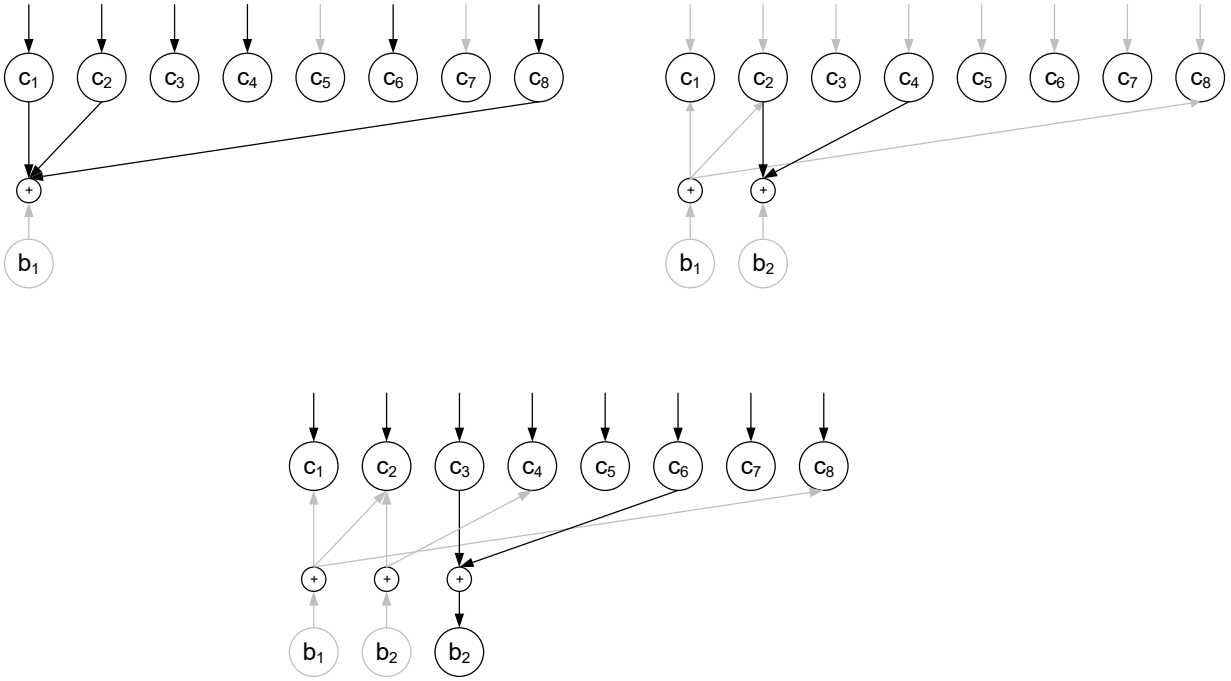

Figure 9: Stages needed for calculation of b3

Note that 9 show in black the connections that must be updated in the stage shown, and in grey those who don't need to be updated.

The complexity of this architecture is rather significant. Each N_b-sized received $\mathbf{c}$ vector may need from N_b demapping operations up to $N_b \times N_b/2$ (half of the decoding matrix $\mathbf{D}$), depending on the complexity of the decoding matrix.

Architecture 2: Demapping of all needed bits for the current b_i

In this architecture there is no need to reset any factor-graph values present in the system. For the calculation of each single b_i, we firstly demap all c_j needed to calculate the current b_i (the relationship is established in the decoding $\mathbf{D}$ matrix). The information updated in the related c_j is combined with all of the previous information that was connected to variable nodes of those c_j, and after combined, is used to calculate directly b_i. After that, a backwards propagation that only affects the nodes directly connected to b_i is performed (the rest of the nodes maintain their previously calculated values).

Architecture 3: Demapping of only the newly needed bits

Once again this architecture does not require any factor-graph value reset between stages, For the calculation of each single b_i, we only demap any needed c_j to b_i **which has never been demapped in previous stages**. The information updated in the related c_j is combined with all of the previous information that was connected to variable nodes of those c_j, and after combined, is used to calculate directly b_i. After that, a backwards propagation that only affects the nodes directly connected to b_i is performed (the rest of the nodes maintain their previously calculated values). Note that in the case of the lower triangular matrix, this

means that only one bit will be processed at each MLC level (just as if there was no random decoding stage).

Architecture 3b: Demapping of only the newly needed bits with iterations

This architecture is the same than the previous one. However, after the calculation of the last pending b_i bit, and with the a priori information obtained, we process once again the first MLC layer (b_0) and continue the process for the next b_is as usual. This architecture proved in unofficial tests to be of great aid to improve the performance of the Binary MMSE Demapper.

2.5 Error-Correcting Encoder and Decoding Stage

2.5.1 Introduction

The purpose of Error-Correcting encoder is to introduce redundant bits, created from the original bits to be transmitted, in the stream of data to be sent. In reception, these redudant bits allow for the correction of transmission errors, and therefore the recovery of the originally transmitted signal.

There are many types of error-correcting schemes. At the fundamental level, they can be divided in block error correcting codes, in which a fixed length of bits is treated independently of the rest of the stream for error-correction redundancy checks, and stream error correcting codes, in which the redundancy is added in between the information bit without clear and defined initial and ending points of the error-correcting scheme.

This stages are typically referenced as (n, k) codes, with n being the length of the resulting error-correcting encoder output, and k being length of the original stream of bits. $r = k/n$ is named the *rate* of the code, and it's a measure of the amount of redundancy present in the output sequence. Higher rates mean lower redundancy, which mean lower capacity to correct errors present in the stream.

For any given SNR and outage percentage, it's important to set specific r rates to meet the required error-correcting capabilities. Note that in our MLC architecture, this will mean individual r_i rates for each of the i-th MLC levels; the rates needed will change depending on the MLC level, SNR and outage allowable. How to allocate specific r_i rates for each level is a subject explained in the Results chapter.

An explanation of how these codes work is beyond the scope of this thesis. In this section, we'll briefly explain the characteristics of the two error correcting systems tested in the simulations.

2.5.2 Perfect Decoder

The perfect decoder is an hypothetical construct which is devised as a decoder which is always capable of correcting all of the errors of the stream, and furthermore, it does so with infinite llrs. This decoder is simulated by simply sending to the random decoding stage a $+\inf$ if the bit to be decoded, b_i, is 1, and $-\inf$ if it was 0.

This decoder is useful as a benchmark for the simulation system, as it negates any loss caused by the limitations of the error-decoding stages, and thereby allows for the calculation of unique losses caused by other stages.

2.5.3 DVB-S2 LDPC Decoder

In order to add to the simulations a real-world, state-of-the-art error-correcting decoder, a DVB-S2 LDPC implementation was added to the simulator [11]. Low Density Parity Check error correcting codes are block codes introduced by Robert G. Gallager [6] and later rescued for usage in methods following the turbo-principle. They are implemented by creating a factor-graph based on the parity check matrix (**H**) of the error correcting stage. This parity check matrix is sparse, and has the property that **Hb=0**, which allows for the creation of the structure of the factor-graph in the form of several xor nodes connected to the variable nodes. Information is passed from the n input variable nodes to the xor check-nodes formed by the parity check matrix, and after a series of iterations the code should converge in the corrected n-sized codeword.

These block codes have a n block length of 64800 bits, and the available rates are 1/4, 1/3, 2/5, 1/2, 3/5, 2/3, 3/4, 4/5, 5/6, 8/9, and 9/10.

3 Results

3.1 Introduction

This chapter illustrates the wide range of simulations performed in order to evaluate the most important alternatives of the system described in . Each of the simulations will be accompanied by a description of the elements that were tested, as well as any pertinent comment regarding the observed results.

All of the executed simulations have a fixed set of restrictions, to which all of them abide. The following list depicts the aforementioned restrictions:

Restrictions

- The channel has independent, identically distributed AWGN noise.
- The fading in the channel is constant for the whole duration of the error-correcting datablock transmission, which means that it's *quasi-static fading*.
- The receiver knowns the channel but the transmitter does not, that is, it doesn't have channel state information (csi).

As one of our main objectives is the evaluation of the spectral efficiency of the designed system, the outage capacity of our system will be main metric. The most promising system designs will then be evaluated for other important metrics such as rates per bit and dispersion plots.

3.2 Performance Metrics Calculations

3.2.1 Introduction

This subsection focuses on explaining how the metrics shown in this section were calculated. In chapter , the mathematical basis for the stages studied in this project was established.

As we described in the , our system works with *soft* information in the form of llrs. This means that at the end of each of the MLC level iterations, the random decoding stage output of the b_i bit that has been processed (see 2.1), in the form of llr, will have to be saved in order to calculate the system's metrics.

From this point onwards, we follow three important steps to convert the llrs into meaningful information:

1. Convert the output llrs of our simulations into mutual information.
2. Convert the mutual information into outage transmission rate.
3. Understand the relationship between this outage transmission rate and the rate per bit that we need for each MLC layer.

This section focuses on these three steps.

3.2.2 LLR to mutual information

In order to analyze the performance of our *soft* transmission schemes, a efficient method for calculating mutual information rates from the *log-likelihood ratios* used in the receiver is needed. This method was described by Stephen ten Brink in [7]. Let X be the random variable that models the $\{-1, +1\}$ transmitted bits. Let L be the random variable that describes the LLRs calculated at the receivers. Note that L, unlike X, is not a discrete random variable and can take any value $\in \{-\infty \leq l \leq +\infty\}$. The equation (65) illustrates the definition of mutual information in this case.

$$I(X, L) = \sum_{\forall x} \int_l p(X = x, L) log_2 \frac{p(X = x, L)}{p(X = x)p(L)} \tag{65}$$

The formula (66) describes the same equation, replacing the joint distributions used in (65) with conditional probabilities.

$$\begin{aligned} p(X = x, L) &= p(X = x)p(L/X = x) \\ p(L) &= \sum_{\forall x} p(X = x)p(L/X = x) \\ I(X, L) &= \sum_{\forall x} p(X = x) \int_l p(L/X = x) log_2 \frac{p(L/X = x)}{\sum_{\forall x} p(X = x)p(L/X = x)} \end{aligned} \tag{66}$$

We assume that $p(X = x) = 0.5$, as before the transmission stage, a compressing algorithm would be used to maximize the entropy of the transmitted bits. As a result, the expression (67) is obtained.

$$I(X, L) = \frac{1}{2} \sum_{x=-1,+1} \int_l p(L/X = x) log_2 \frac{p(L/X = x)}{\frac{1}{2}p(L/X = -1) + \frac{1}{2}p(L/X = +1)} \tag{67}$$

In [7] it was shown that $P(L/X = x)$ can be described as a normal distribution, depicted in (68).

$$\mu_L = \frac{\sigma^{2_L}}{2}$$
$$p(L/X = x) = \frac{1}{\sqrt{2\pi\sigma_L^2}} e^{-\frac{(L-\mu_L x)^2)}{2\sigma_L^2}} \tag{68}$$

Replacing the $P(L/X = x)$ in (67) with the normal distributions shown in (68) results in the equation (69).

$$I(X, L) = 1 - \int_l \frac{1}{\sqrt{2\pi\sigma_L^2}} e^{-\frac{(L-\sigma_L^2/2)^2}{2\sigma_L^2}} log_2(1 + e^{-L}) dL \tag{69}$$

The equation 69 can be further simplified if we consider that we've a gaussian distribution of the form $\mathcal{N}(\sigma_L^2/2, \sigma_L^2)$ and we approximate the result:

$$\begin{aligned} I(X, L) &= 1 - \int_l \frac{1}{\sqrt{2\pi\sigma_L^2}} e^{-\frac{(L-\sigma_L^2/2)^2}{2\sigma_L^2}} log_2(1 + e^{-L}) dL \\ &= E\{1 - log(1 + e^{-Lx})\} \approx \frac{1}{N} \sum_{n=1}^{N} (1 - log(1 + e^{-L_n x_n})) \end{aligned} \tag{70}$$

It's the last expression in 70 the one we'll use to calculate the mutual information for each b_i bit, by using it's corresponding calculated llr value L_i alongside it's real transmitted binary value x_i.

3.2.3 Mutual Information to transmission rate and outage transmission rate

Shannon defined the capacity of a channel as the maximum mutual information between the received signal and the transmitted one that can be consistently attained in that channel [9]:

$$C = max(I(y, x)) \tag{71}$$

Where **y** would be the received signal, in phase and amplitude, and **x**, in our case, would be the transmitted bits. One important property of MLC system such as the one used here is that we can apply the chain rule to calculate the total mutual information taking into account each decoded bit b_i [8] [3]:

$$I(\mathbf{y}; \mathbf{b}) = I(\mathbf{y}; b_0) + I(\mathbf{y}; b_1/b_0) + I(\mathbf{y}; b_2/b_0 b_1) + \ldots + I(\mathbf{y}; b_{N_b}/b_0 b_1 \ldots b_{N_b-1}) \tag{72}$$

Given the MLC scheme, after the calculation of each output bit b_i, and applying the expression developed in 3.2.2, we actually will obtain one term of the equation 72. For example, the llr that we will obtain after b_0 and b_1 have already been processed, and we've just calculated b_2, will, after converting it to mutual information, be exactly $I(\mathbf{y}; b_2/b_0b_1)$. This indicates that it's possible to simply use the llrs calculated at each output b_i, convert them to mutual information, and add all the N_b terms in order to obtain the total mutual information and hence the transmission rate attained, which ideally would be near the aforementioned capacity of the channel.

Therefore, the transmission rate will be calculated as shown in 73 taking into account that we can apply the chain rule shown in 72 to solve it.

$$\text{Transmission Rate} = I(\mathbf{y}; \mathbf{b}) \tag{73}$$

However, as we mentioned in the thesis introduction, this transmission rate is not truly useful in the case of *quasi-static fading* channels. This is due to the fact that, as the fading is constant, and could potentially be a deep fade, during the whole transmission of the datablock, the rate needed to decode that datablock can vary widely depending on the channel. Therefore, these channels are more properly measured by what is called the outage probability: the probability that the rate needed to properly decode the datablock is smaller than a set rate that we'll use in the system, and therefore, that we lose the information in that datablock.

In our tests we'll speak of the outage capacity and outage transmission rate: the maximum capacity attainable, and transmission rate attained for a given set of channels and a given outage probability.

The implemented software has functions to calculate the outage transmission rate given an outage probability (typically 10 percent in our tests) and the transmission rate calculated in 73. This software calculates the outage transmission rates per bit, which can then be added in order to obtain the total outage transmission rate.

This calculation is done following this procedure:

1. A set of $1 \times N_b$ sized vectors with tentative rates for each of the N_b bits used is generated. This set of vectors is generated with coarse spacing between them.

2. For each of the evaluated channels in the simulation, the transmission rate (73) is subtracted from the vectors generated in 1.

3. The vector which has the maximum rate, but which still after being subtracted the transmission rates in 2 has a number of channels in outage lower than the set outage probability is used as the seed for the new iteration.

4. A new set of reference vectors is created from the vector chosen in 3, but each iteration with less coarse spacing.

After a few iterations, a vector of rates is found which provides the maximum transmission rate possible if we can only afford a given outage probability.

On the other hand, we'll also need the reference gaussian capacity of the system. This capacity, for a MIMO channel without channel state information can be calculated as [2][10]:

$$C = log_2(det(\mathbf{I} + \frac{SNR}{N_t}\mathbf{H}\mathbf{H}^H)) \tag{74}$$

Where **H** will be each of the channels that we've evaluated in the simulation. We will calculate one capacity for each channel. Now, once again these capacities are not realistic unless we apply the same outage probability principles we've applied to the transmission rate. In order to do so, the software does a task identical to the one described to calculate the outage transmission rate, in the preceding paragraphs.

3.2.4 Rate selection for error-correcting codes

This subsection will try to clarify how to select the error-correction stage rates needed for each bit. Fortunately, the MLC architecture provides an easy and powerful way to determine these rates: the information chain rule seen in 72. Once the transmission rates for each bit have been calculated with the chain rule, and the outage transmission rate derived from this information, each converted component ($I(y; b/b_0...)$) will be the target rate to which we will have to abide. We can see this graphically in the outage rate per bit graphs that will be shown in the most relevant test in this chapter.

3.3 Random Encoding Stage Elements Tests

3.3.1 Introduction

In this section, we'll test the most significant different alternative methods for the random encoding and decoding stage, as described in 2.4:

- Decoding architecture as described in 2.4.
- Lower triangular matrix vs upper triangular matrix
- No interleaver, Spatial Interleaver, Intrablock Temporal Spatial Smart Interleaving, Intrablock Temporal Spatial Interleaving, Intrablock Temporal Interleaving.

All of the test in this subsection will be done using a perfect error correcting stage and a MAP demapper, in order to isolate the losses that are generated exclusively by this stage.

Finally, a in-depth analysis will be done of the best methods, against other known and meaningful methods.

3.3.2 Random Decoder Architecture 3

This architecture, for any given output MLC level (bit b_i) that must be decoded, only demaps the new c_j (random coded) bits that have never been demapped until this point. For those bits, it does indeed take into account all of the *a priori* information available until now. More information is available in 2.4.

Test Configuration

MIMO Setup	2x2
Simulated Channels	100
Simulated Timeslots per Channel	64800
Constellation and labeling in use	16 QAM ADD
Demapper type	MAP
Error-correcting stage type	Perfect Decoder
Random Decoder Type	Factor Graph
Random Decoder Architecture	Type 3 to be tested
Random Decoder Matrix Type	[Lower,Upper] triangular to be tested
Random Decoder Interleaver	All types to be tested

Reference plots used

Outage Capacity for 10 percent outage probability
16 QAM Capacity for 10 percent outage probability
10 purerandom codes with lock-up tables demapping
Standard MLC, without Random Encoding or Decoding

Note that the "10 pure random codes with lock-up tables demapping" are the same than the ones that were showcased in Lamarca's paper [1], and it's purpose it to serve as benchmark. Remember that as explained in the introduction, the lock-up tables implementation used in that paper cannot be used for large numbers of N_b.

Caption 11 shows the results for all interleavers and architecture 3 in the case of the lower triangular encoding or decoding matrix.

As can be seen in the caption, the performance of the lower triangular generator and decoding matrices system highly depends on the use of spatial interleaving. This phenomenon has a simple answer. First of all, 75 shows the Decoding matrix of a generic lower triangular code. Remember that the expression that links the random decoder output bits **b** and the input bits **c** is **b**=**Dc**.

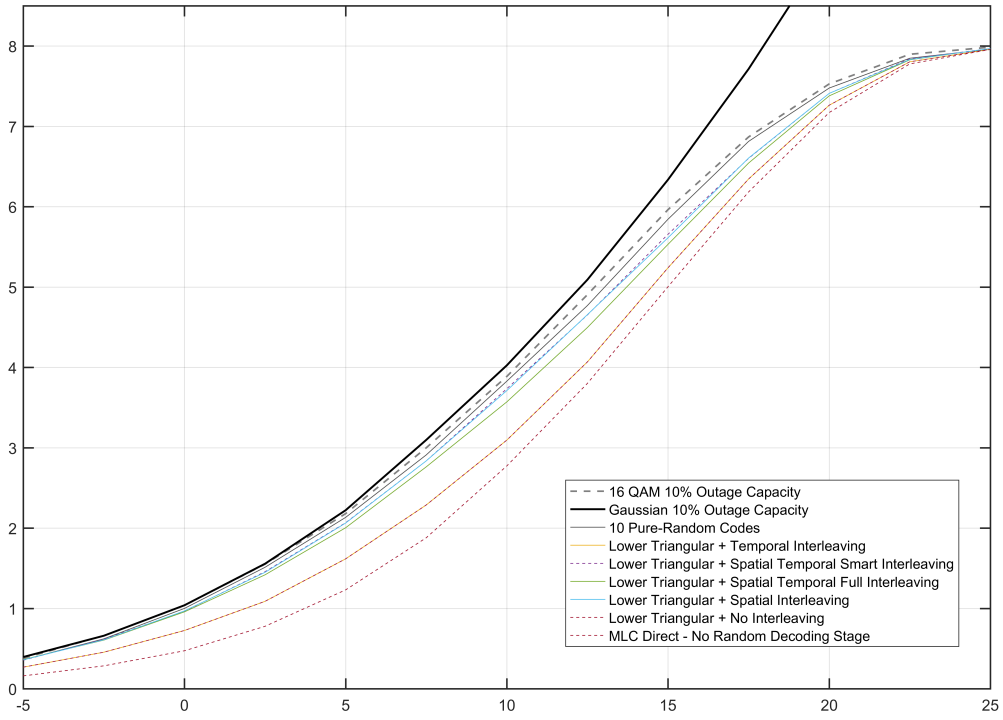

Figure 10: Architecture 3 and Lower-Triangular Matrices

$$\mathbf{D} = \begin{bmatrix} 1 & 0 & 0 & \dots & 0 \\ d_{10} & 1 & 0 & \dots & 0 \\ d_{20} & d_{21} & 1 & \dots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ d_{N_b 0} & d_{N_b 1} & d_{N_b 2} & \dots & 1 \end{bmatrix} \tag{75}$$

If we watch carefully at the structure of these matrices, we can see that b_1 always depend on c_1 alone, that is, they're equal. If we don't do spatial interleaving, there is a good chance that the constellation labeling bit at which we're transmitting c_1 has bigger or smaller attenuation depending on the channel realization. By doing spatial interleaving, we help this kind of codes a lot to smooth out these kind of problems, as c_1 is transmitted on different constellation labeling bits every time and therefore the capacity of that bit is averaged over time.

On the other hand, both the spatial interleaving and spatial temporal interleaving methods work remarkably well. They approach the transmission rates of the 10 pure random codes with a very simple approach (architecture 3), which only requires to demap once each c_i bit.

Regarding upper triangular matrices, in 11 it's possible to see the results that they provide with this architecture.

As seen in the aforementioned figure, it becomes clear that the upper triangular encoders do not fare well with this kind of simplistic architecture. With all interleavers, at some point the

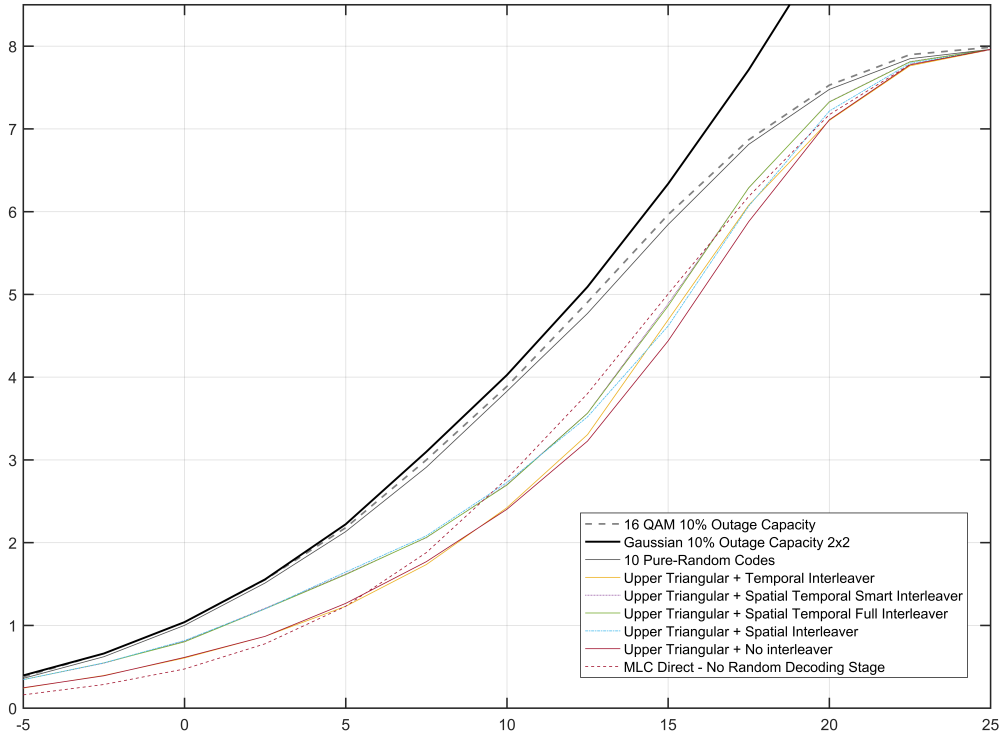

Figure 11: Architecture 3 and Upper-Triangular Matrices

transmission rates of the codes are bellow the MLC standard, non-random encoder equipped system. The explanation for this is due to the fact that these codes, for the calculation of each b_i bit, *always* rely on sets of c_j bits of which most of them are not known completely by the decoder after performing the error correction stages. This is a stark contrast with the lower triangular codes: 75 shows that after every MLC level decoding, we already know all of the previous c_j bits from which the next b_i bit may depend, with the exception of a single one which is new. This is exactly the same situation we have in standard MLC architecture (without random enconding or decoding).

Moreover, in the check nodes used in the factor-graph of the random decoder, the llr output from the node can be approximated as the minimum llr among all the inputs. This means that a greater number of uncertain bits, as is the case of the upper triangular matrices and specially in the case that we only demap the new, never demapped bits at each iteration, always produces a poorer output. Therefore in this situations it's very important that the input bits c_j have as much information as possible before calculating the corresponding b_i.

Finally, in 12, a comparison between the best results of the upper and lower triangular matrices can be seen.

The result points a clear victory of the lower triangular matrices over the upper triangular matrices in this case.

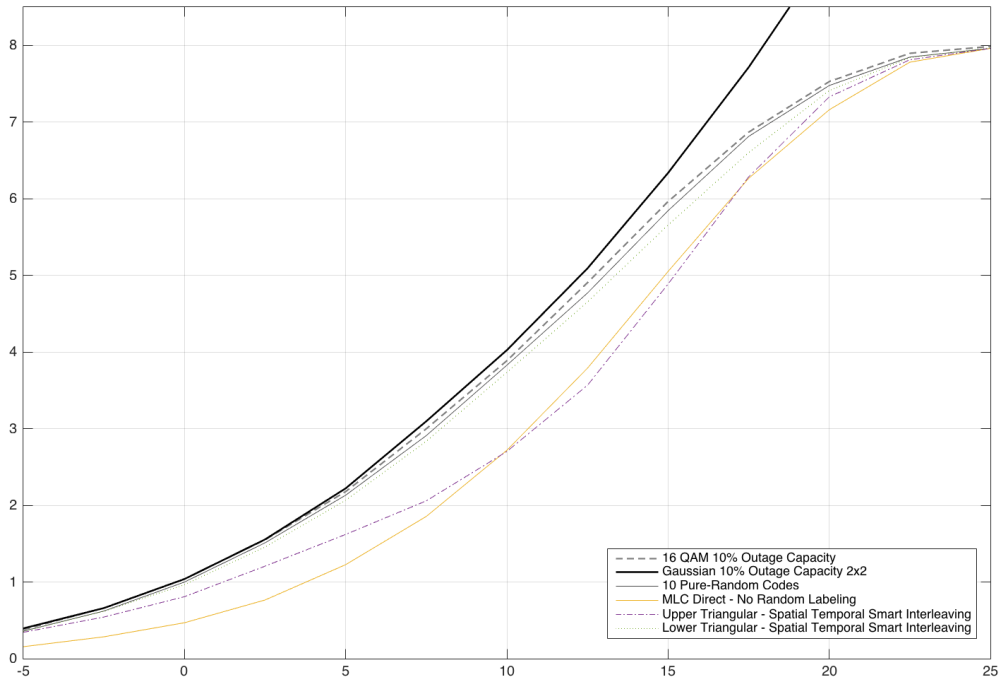

Figure 12: Architecture 3 - Best Upper and Lower Triangular Matrices Methods

3.3.3 Random Decoder Architecture 2

In this demapper architecture, for any given output MLC level (bit b_i) that must be decoded, the system only demaps the c_j (random coded) bits that are strictly needed for random decoding b_i. For those bits, it does indeed take into account all of the *a priori* information available until now. More information is available in 2.4.

Test Configuration

MIMO Setup	2x2
Simulated Channels	100
Simulated Timeslots per Channel	64800
Constellation and labeling in use	16 QAM ADD
Demapper type	MAP
Error-correcting stage type	Perfect Decoder
Random Decoder Type	Factor Graph
Random Decoder Architecture	Type 2 to be tested
Random Decoder Matrix Type	[Lower,Upper] triangular to be tested
Random Decoder Interleaver	All types to be tested

Reference plots used

Outage Capacity for 10 percent outage probability
16 QAM Capacity for 10 percent outage probability
10 purerandom codes with lock-up tables demapping
Standard MLC, without Random Encoding or Decoding

Note that the "10 purerandom codes with lock-up tables demapping" are the same than the ones that were showcased in Lamarca's paper [1], and it's purpose it to serve as benchmark. Remember that as explained in the introduction, the lock-up tables implementation used in that paper cannot be used for large numbers of N_b.

Once again, figure 14 shows the results for all interleavers and architecture 2 in the case of the lower triangular encoding or decoding matrix.

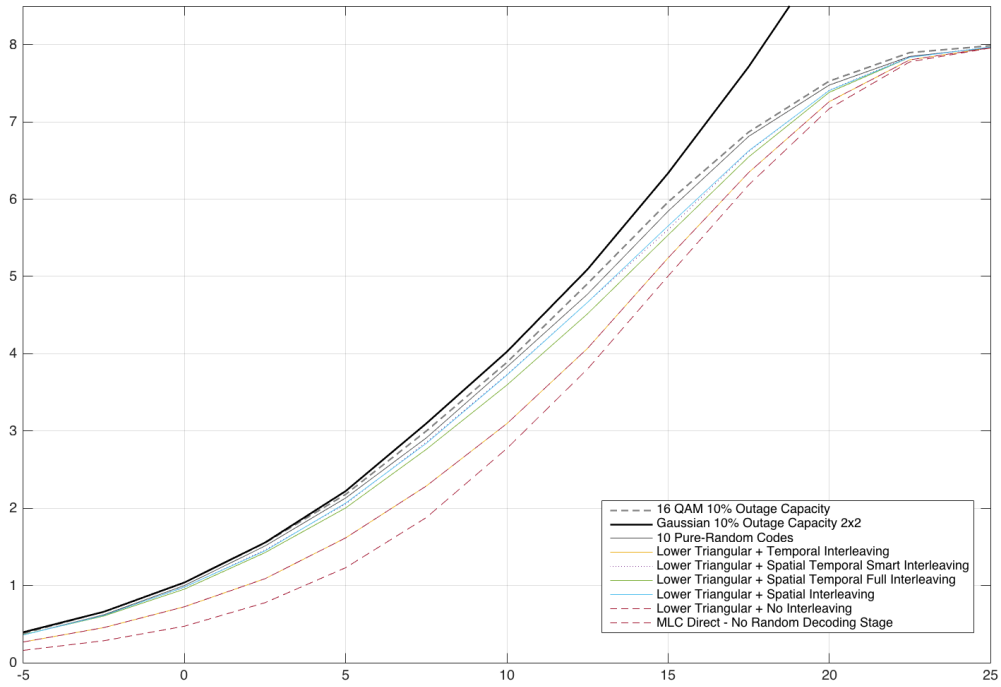

Figure 13: Architecture 2 and Lower-Triangular Matrices

As can be seen in the figure, the performance of the lower triangular generator and decoding matrices system also depends on the use of spatial interleaving. The reasons for this are the same than the ones described in the "Architecture 3" subsection.

The rates attained seem to be very similar to the ones found in the Architecture 3 results. This may be due to the aforementioned fact that the lower triangular systems never depend on more than 1 non-error-correcting decoded bit. If we need to decode, for example, b_3, due to the decoding matrix itself, we can be sure that we've already error-corrected llrs for c_0 to c_2, which tend to be far better than the llrs provided directly from the demapper (such as c_3), in the previous example.

On the other hand, regarding upper triangular matrices, in 14 it's possible to see the results that they provide with this architecture.

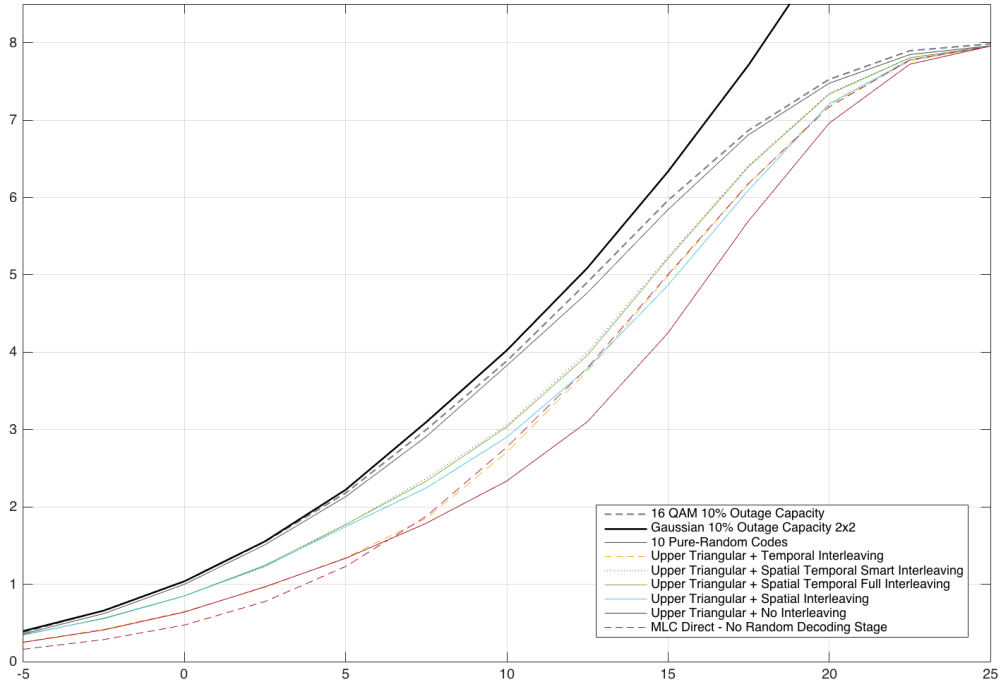

Figure 14: Architecture 2 and Upper-Triangular Matrices

In these plots it's possible to see that there is significant gains from, in order to decode b_i, demap in this stage all the needed c_j with the latest *a prioris*. But these results are still far from optimal. Furthermore, we can see one further problem that strongly affects the methods which do not involve intra-block temporal interleaving: there are losses due to what we called, in 2.4.2, inter-stage cycles. These are due to the fact that, in these situations, the different c_js used for decoding a given output vector $\mathbf{b}$ belong to the same transmitted symbol, and therefore are not entirely uncorrelated.

Finally, in 15, a comparison between the best results of the upper and lower triangular matrices can be seen.

Once again, the lower triangular matrices attain greater performances than the upper triangular matrices methods.

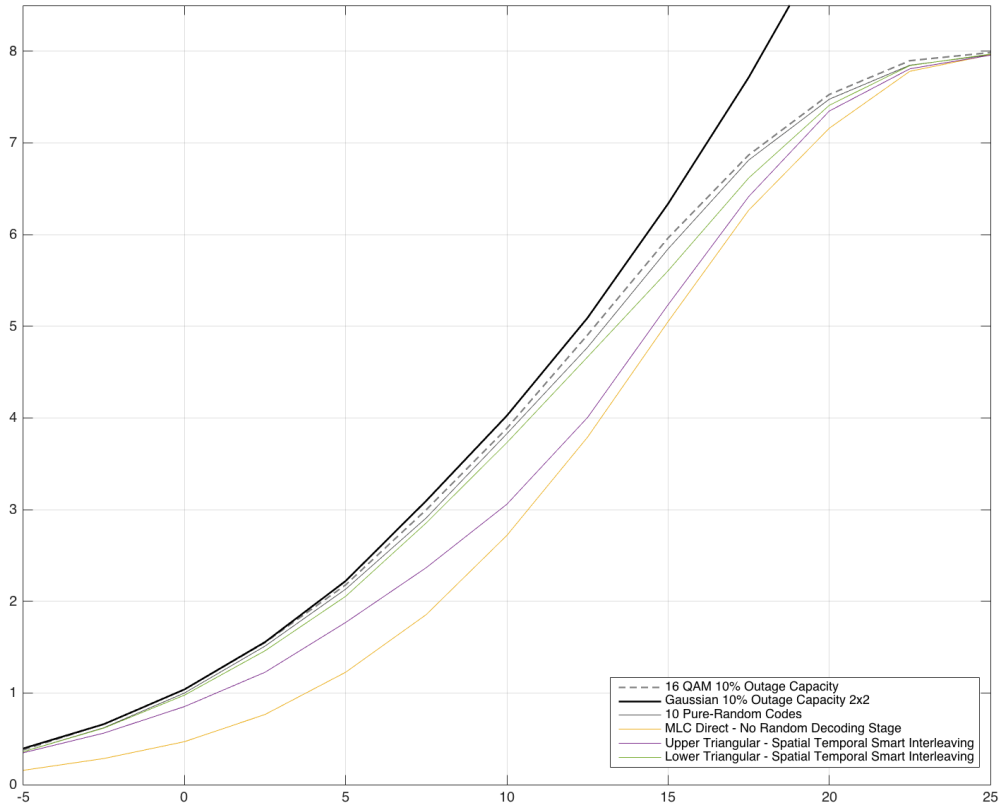

Figure 15: Architecture 2 - Best Upper and Lower Triangular Matrices Methods

3.3.4 Random Decoder Architecture 1

This architecture, for any given output MLC level (bit b_i) that must be decoded, we reset the factor graph values to 0, with the exception of the b_i variable nodes, and demap all previously demapped bits (up to the ones needed for b_i) with the new a priori information. Then the MLC sequence is followed in order (ie: starting with the calculation of b_0, although no new information is sent to the error-correcting stage nor updated from it). For further information about this architecture please see 2.4.

Test Configuration

MIMO Setup	2x2
Simulated Channels	100
Simulated Timeslots per Channel	64800
Constellation and labeling in use	16 QAM ADD
Demapper type	MAP
Error-correcting stage type	Perfect Decoder
Random Decoder Type	Factor Graph
Random Decoder Architecture	Type 1 to be tested
Random Decoder Matrix Type	[Lower,Upper] triangular to be tested
Random Decoder Interleaver	All types to be tested

Reference plots used

Outage Capacity for 10 percent outage probability
16 QAM Capacity for 10 percent outage probability
10 purerandom codes with lock-up tables demapping
Standard MLC, without Random Encoding or Decoding

Note that the "10 purerandom codes with lock-up tables demapping" are the same than the ones that were showcased in Lamarca's paper [1], and it's purpose it to serve as benchmark. Remember that as explained in the introduction, the lock-up tables implementation used in that paper cannot be used for large numbers of N_b.

Figure 17 shows the results for all interleavers and architecture 1 in the case of the lower triangular encoding or decoding matrix.

The figure shows results very similar to those obtained in the Architecture 3 section, the main difference being that the system complexity in this case is much higher as we need to demap something in between N_b and $N_b \times N_b/2$ bits per each antenna transmission that we want to decode.

Regarding upper triangular matrices, in 17 it's possible to see the results that they provide with this architecture.

In these plots it's possible to see that there is once again significant gains from architectures 2 and 3. One again, as explained in the architecture 2 results, we can see the devastating effects of the cycles in the non temporary interleaved methods.

Finally, in 18, a comparison between the best results of the upper and lower triangular matrices can be seen.

This last test still shows an advantage of the lower triangular matrices over the upper triangular ones. One of the main reasons could be that, while with intra-block temporal demapping

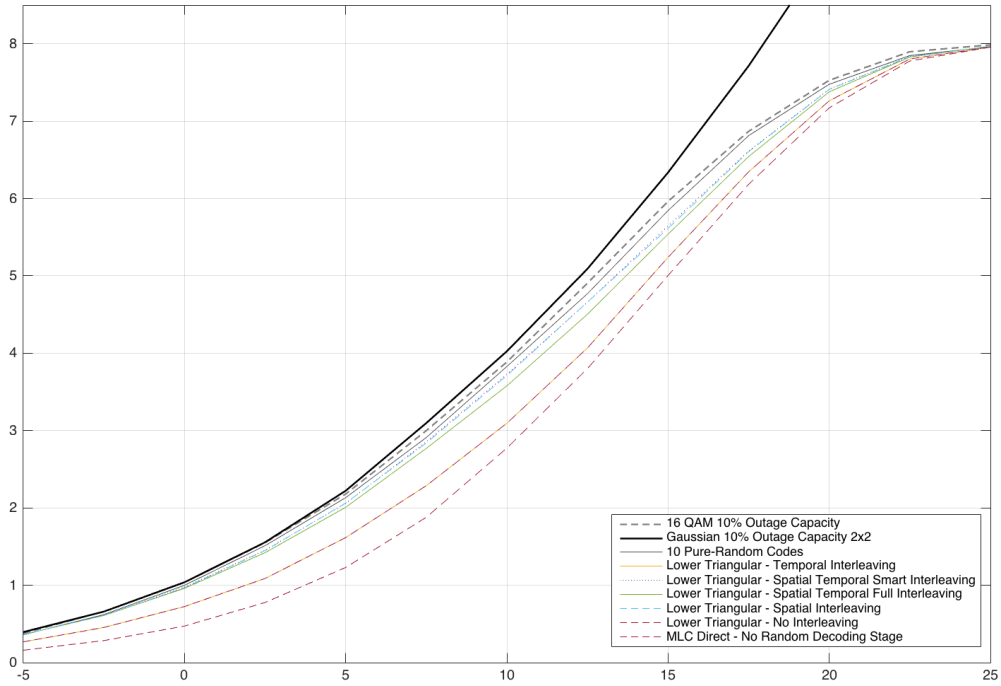

Figure 16: Architecture 1 and Lower-Triangular Matrices

we enlarge the cycle lengths, this enlargement do not mean that the cycles do no not exist; just that the impact is smaller. Lower triangular matrices could be much more resilient to this phenomenon due to an advantage already mentioned: for each bit to be decoded b_i, any c_j that could be used except one have already been error-corrected by the decoder, and therefore it's llrs are more deterministic and less prone to information losses due to cycles.

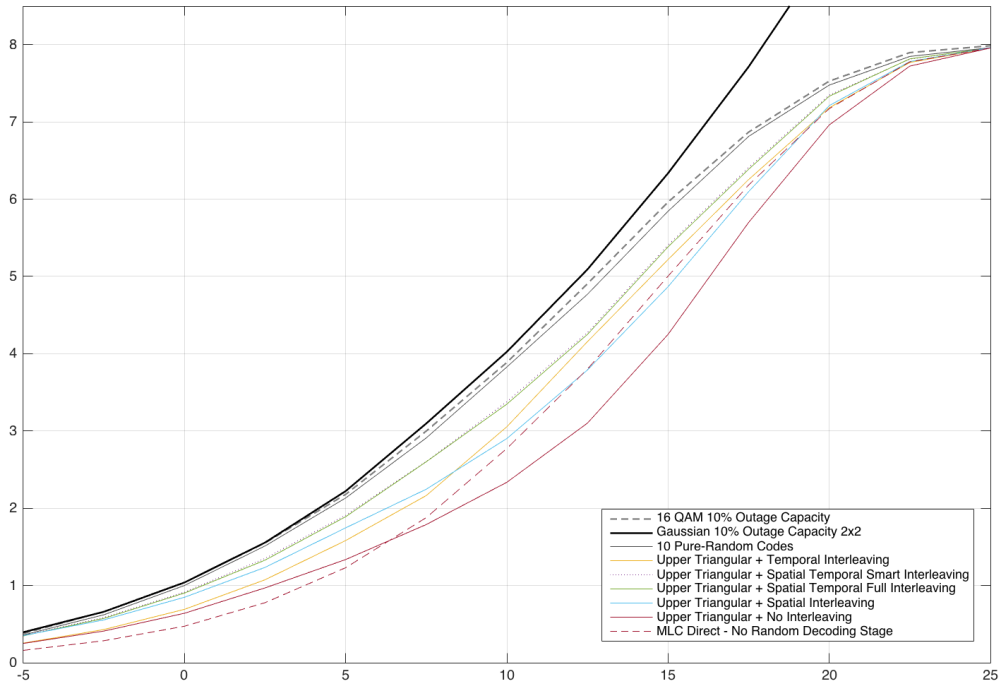

Figure 17: Architecture 1 and Upper-Triangular Matrices

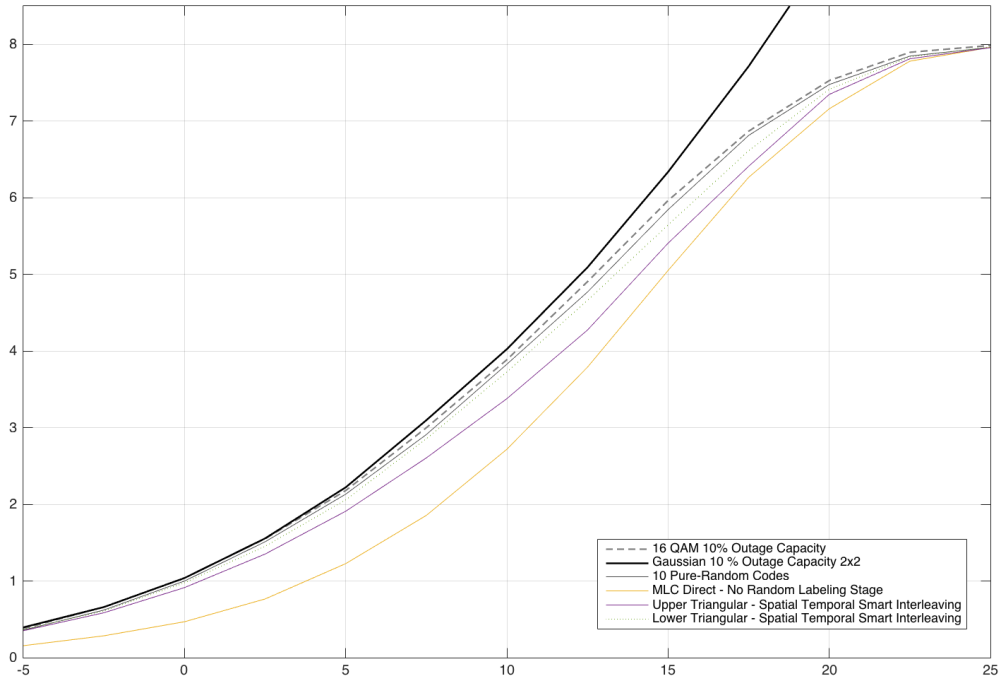

Figure 18: Architecture 1 - Best Upper and Lower Triangular Matrices Methods

3.3.5 Comparison of the best methods of architectures 1, 2 and 3

Finally, figure 19 shows all the best Upper and Lower Triangular Methods of all 3 architectures.

Test Configuration

MIMO Setup	2x2
Simulated Channels	100
Simulated Timeslots per Channel	64800
Constellation and labeling in use	16 QAM ADD
Demapper type	MAP
Error-correcting stage type	Perfect Decoder
Random Decoder Type	Factor Graph
Random Decoder Architecture	All three types to be tested
Random Decoder Matrix Type	[Lower,Upper] triangular to be tested
Random Decoder Interleaver	Temporal Spatial Smart Interleaver

Reference plots used

Outage Capacity for 10 percent outage probability
16 QAM Capacity for 10 percent outage probability
10 purerandom codes with lock-up tables demapping
Standard MLC, without Random Encoding or Decoding

Two important conclusions can be extracted from this graph, firstly that as seen in the previous test lower triangular matrices perform better than upper triangular ones, and secondly, and most important, lower triangular matrices have little to no impact to the fact whether they use the architecture 1, 2 or 3. **This means that the architecture 3, much more simple with merely N_b demappings needed for each N_b random decoded words, is the best solution overall.** Regarding the interleaver, given that the complexity added by using the Spatial Temporal Smart Interleaver is very small, and that it has proven that can reduce the effects of cycles, we consider that in real life scenarios the interleaver is preferable over the other ones.

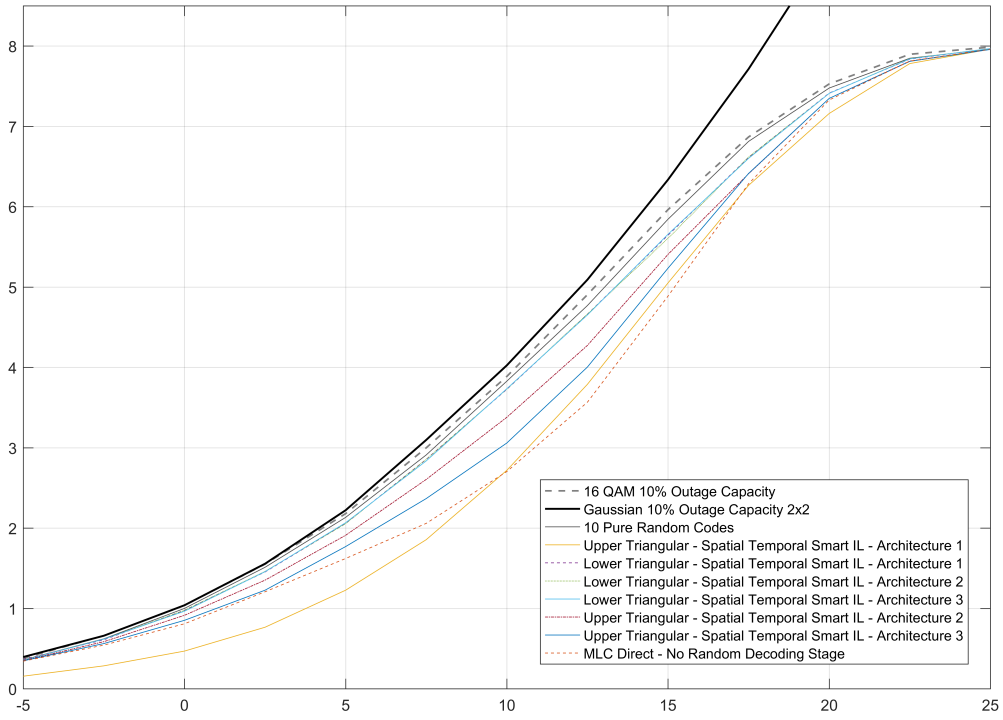

Figure 19: Best Upper and Lower Triangular Methods of all 3 architectures

3.3.6 Analysis of the best method

From the previous subsections it can be derived that the best method was the lower triangular matrices one alongside architecture 3 and spatial temporal smart interleaver.

In this section, we'll analyze its performance in terms of rates per bit as well as dispersion, and compare it to the reference used in the previous sections: "10 pure random codes", such as the ones used in the paper which was the basis for this thesis [1]. In 20 we can see the bitrates per code in the Lower Triangular with Spatial Temporal Smart Interleaving in architecture 3 method, while in 21 we see the ones of the reference 10 pure random codes.

Comparing both figures there is one obvious conclusion: **the principles of operation of the lower triangular linear codes and MATLAB's random labelings are quite different**. Both of them are relatively close in terms of final spectral efficiency, but while the first one in general needs per bit rates quite close to each other in one specific SNR level, the second one tends to spread the needed rates all along this SNR.

This is in fact a useful, non-expected property of this system: it tends to need higher starting (ie. for the first MLC levels) rates than the random codes, which is something that it's much easier to attain with the present error-correcting codes than rates that tend to approach zero as in the case of the pure random codes.

On the other hand, the figure 22 shows the dispersions of the transmission rates per bit (x

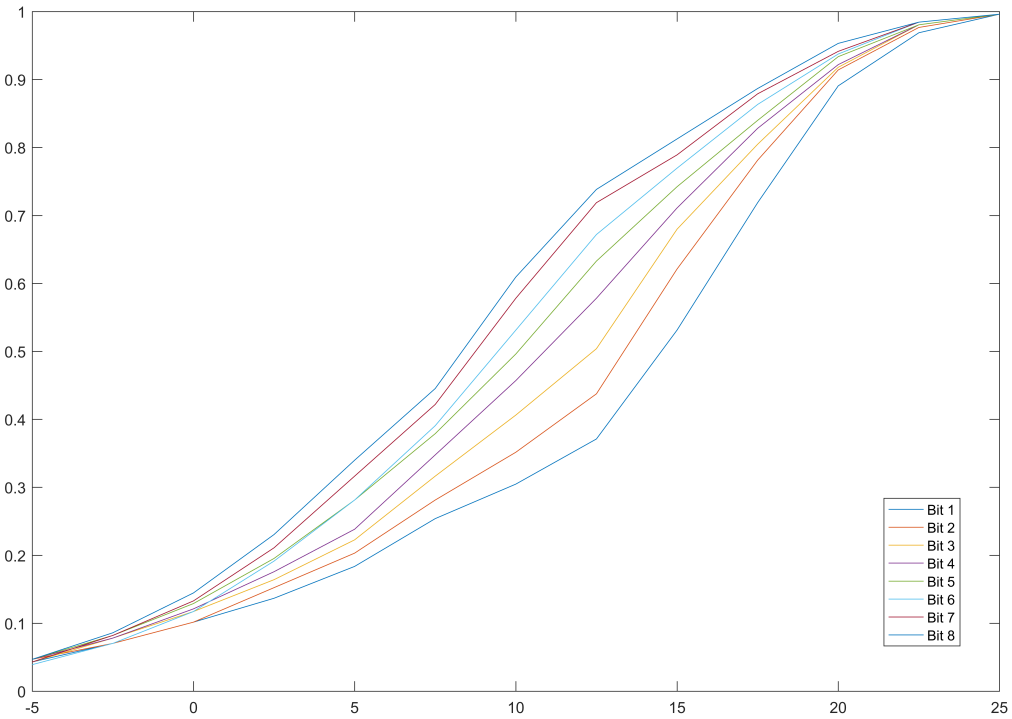

Figure 20: Bit Rate for Lower Triangular with Spatial Temporal Smart Interleaving in architecture 3

axis) vs the total channel transmission rates (y axis). There are a total of 100 channels, each one of them represent one point.

The picture also points to a different way of operation between both systems. **While the pure random codes tend to create a S shaped curve with minimal amounts of dispersion, lower triangular codes tend to create a straight line**, with minor dispersion in the first and last bits. This dispersion is understandable, as in lower triangular systems the first decoded bit, b_0, only depends on a coded bit c_0, which despite being continuously mapped from one constellation labeling bit to another (spatial interleaving), it's still not enough to "take" enough information from other bits, as in the pure random case. It's worth mentioning that in pure random cases it's impossible to ascertain the value of a given coded bit c_i without knowing the entire **b** vector that generated it, which is a further testament on how the original information is spread on all the **c** transmitted bits.

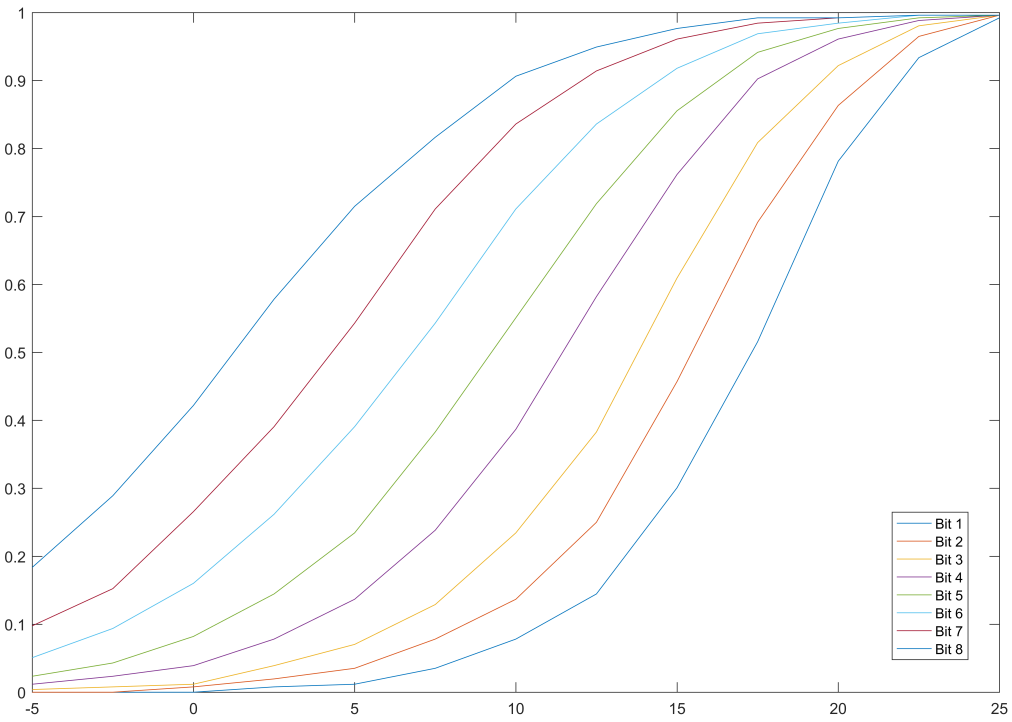

Figure 21: Bit Rate for 10 Pure Random Codes

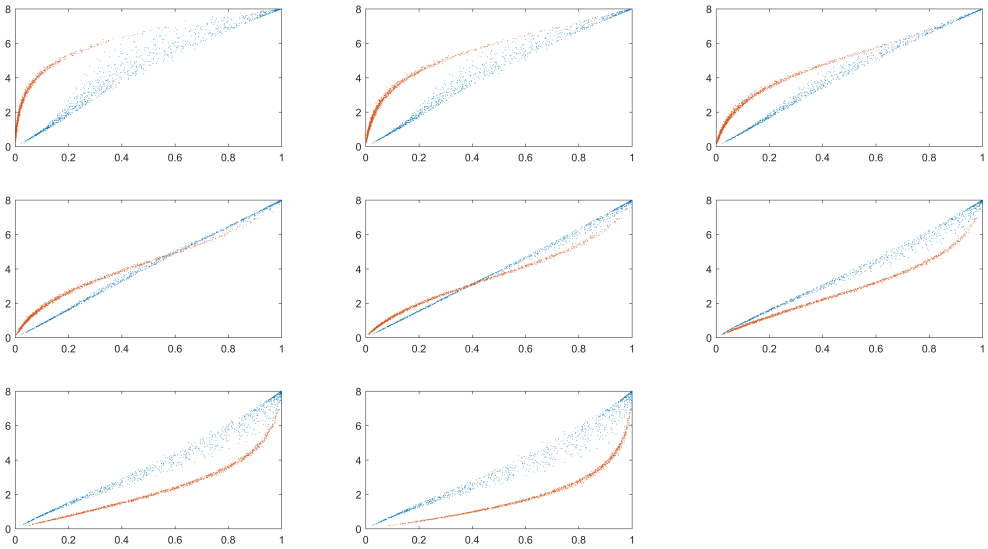

Figure 22: Transmission rate per bit vs Total channel transmission rate dispersion. In blue, triangular lineal codes, in orange, 10 pure random codes

3.4 Binary MMSE Demapper Tests

The MMSE Demapper, explained in 2.3.5, is a computationally very simple demapper that was developed as part of this thesis. One of it main objectives is to be capable of decoding an arbitrary amount of N_b bits, with it's complexity growing in a non-exponential manner.

The simplicity of the demapper puts some limitations on it's performance. In any case, in other works [12] it has been proven that this kind of demapper can reach very good spectral efficiencies so long iterations are executed between the decoder and the demapper. This last point is something that is out of the scope of this thesis.

3.4.1 Binary MMSE Demapper vs MAP Demapper

This section will show the rather important losses that we face if we want to use an extremely simple demapper such as this one.

Test Configuration

MIMO Setup	2x2
Simulated Channels	100
Simulated Timeslots per Channel	64800
Constellation and labeling in use	16 QAM ADD
Demapper type	MAP and Standard MMSE Tested
Error-correcting stage type	Perfect Decoder
Random Decoder Type	Factor Graph
Random Decoder Architecture	Architecture 3
Random Decoder Matrix Type	Lower triangular
Random Decoder Interleaver	Temporal Spatial Smart Interleaver

Reference plots used

Outage Capacity for 10 percent outage probability
16 QAM Capacity for 10 percent outage probability

The figure 23 shows the results of this test. As can be seen, the binary MMSE demapper cannot keep up with the MAP demapper in a single iteration. Other researches [12] show that these type of demappers greatly benefit from iterations, and therefore MLC iterations could be attempted in a future investigation.

Nevertheless, the results are fairly good if we consider that one the one hand we have a MAP demapper whose complexity grows at 2^{N_b} given the size of N_b, while this demapper at most increases its complexity at $N_b \times N_b$, being quite possible to reduce that complexity through optimizations.

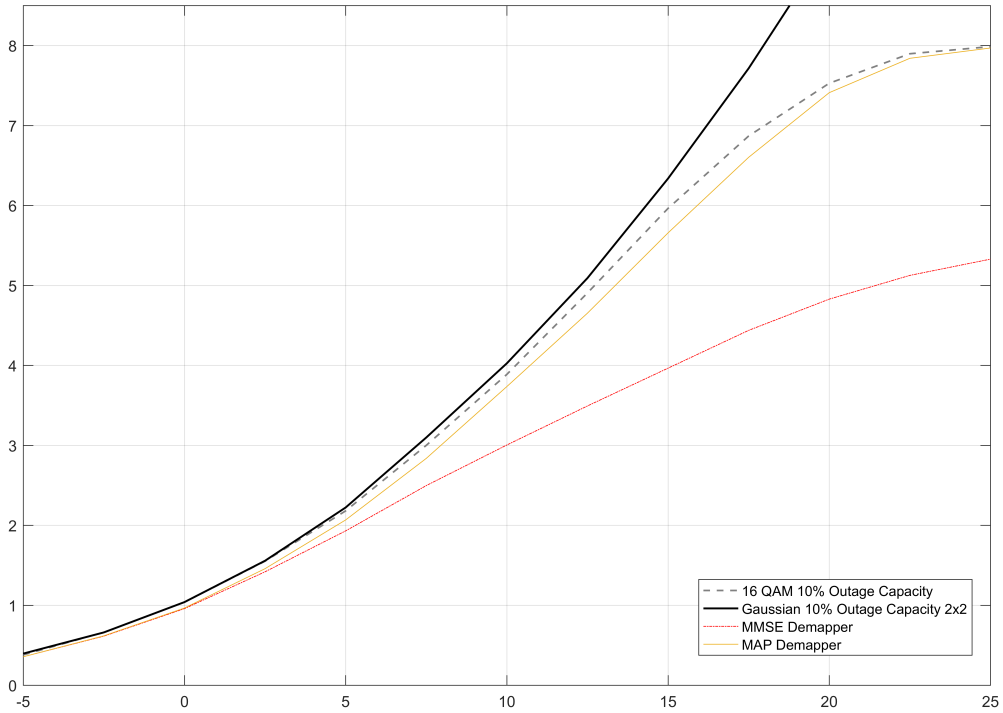

Figure 23: MMSE Demapper vs MAP Demapper for Lower Triangular Matrices with Spatial Temporal Smart Interleaving

3.4.2 Standard Binary MMSE Demapper vs Variant Binary MMSE Demapper

In section 2.3.5 it was explained that two different MMSE Demappers were developed at one stage:

1. Standard Binary MMSE: That uses the probabilities of c_j in order to calculate the covariances and means it needs for its calculation.

2. Variant Binary MMSE: That uses the probabilities of c_j, and, by using the decoding matrix D, calculated the probabilities of b_i so that it can calculate the covariances and means it needs for its calculation in terms of b_i

The second method was expected to be necessary in order to avoid the aforementioned inter-stage cycles: information losses due to the fact that the bits in the same transmission ($\mathbf{c}$ vector) are not statistically independent, and therefore, if there is no posterior intrablock temporal interleaving they would reduce the transmission rate.

Test Configuration

MIMO Setup	2x2
Simulated Channels	100
Simulated Timeslots per Channel	64800
Constellation and labeling in use	16 QAM ADD
Demapper type	Standard MMSE and Variant MMSE Tested
Error-correcting stage type	Perfect Decoder
Random Decoder Type	Factor Graph
Random Decoder Architecture	Architecture 3
Random Decoder Matrix Type	Lower triangular
Random Decoder Interleaver	No Interleaver and Temporal Interleaver

Reference plots used

Outage Capacity for 10 percent outage probability
16 QAM Capacity for 10 percent outage probability

The plot 24 shows the results of these tests. These plots have interesting repercussions. First of all, it's possible to see that the curve is different from the one shown in 23. This is due to several factors:

1. In this case, no MMSE system uses spatial interleaving, which lower triangular matrices specially need due to the reasons stated in the random decoding tests, and therefore we can see the losses due to the lack of resilience against outage probabilities.

2. The transmission rate attained for higher SNRs is higher than the one in 23. This, counter-intuitively is also due to the lack of spatial interleaving. Non-iterating MMSE demappers depend a lot on which is the sequence of bits that are demapped [2]; ideally, the bits with the highest signal power should be demapped first, and then continue until the one with the least power is demapped. When we do spatial interleaving, such as in the 23, this is certainly not the case. With no spatial interleaving, the first two bits of each antenna are the ones with the greatest signal power in our constellations, which means that at least the first two bits demapped by the MMSE follow more or less this rule.

Note how it would seem possible to use a demapper with spatial interleaving for low SNRS 23 while for high SNR, if iterations are undesirable, it would be better to not use interleaving as shown in 24.

Finally, it's worth mentioning that once we introduce somekind of temporal demapping, and therefore enlarge the inter-stage cycle-length of the demapping-random decoding bits c_i, the standard MMSE performs as the variant MMSE, but with a much simpler algorithm. Therefore, by default this is the method used in the next tests.

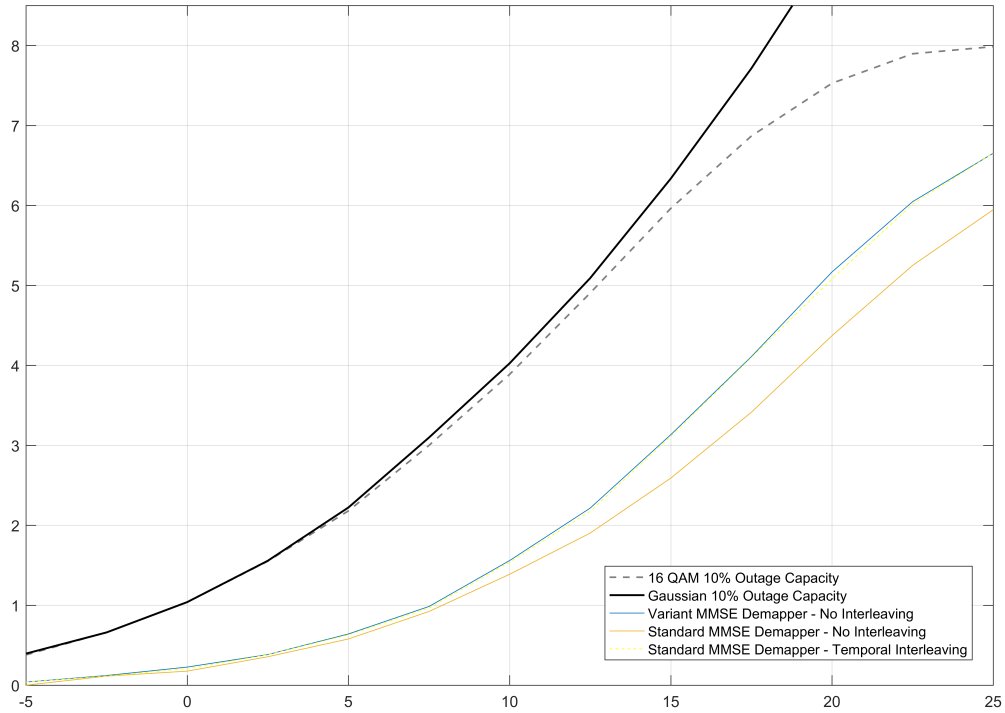

Figure 24: MMSE Standard Demapper vs MMSE Variant for Lower Triangular Matrices with Spatial Temporal Smart Interleaving. No Interleaving and Temporal Interleaving

3.5 Constellation Labeling Performance

Finally, we're going to test that the system is not dependent on the constellation labeling chosen to deliver good performances. In theory, MLC architectures can achieve the capacity for any labeling [3], but as this case could in a certain sense not be considered pure MLC, as the bit output by the demapper is not enough to calculate the bit at the output of the random decoder.

Test Configuration

MIMO Setup	2x2
Simulated Channels	100
Simulated Timeslots per Channel	64800
Constellation and labeling in use	16 QAM ADD and Grey
Demapper type	MAP
Error-correcting stage type	Perfect Decoder
Random Decoder Type	Factor Graph
Random Decoder Architecture	Architecture 3
Random Decoder Matrix Type	Lower triangular
Random Decoder Interleaver	Spatial Temporal Smart Interleaver

Reference plots used

Outage Capacity for 10 percent outage probability
16 QAM Capacity for 10 percent outage probability
10 Pure Random Codes with ADD labellings
10 Pure Random Codes with GREY labellings

Figure 25 show the results of this test. As was expected, no difference whatsoever was observed in either the 10 Pure Random codes or the lower triangular matrix when changing the antenna labellings from ADD to GREY.

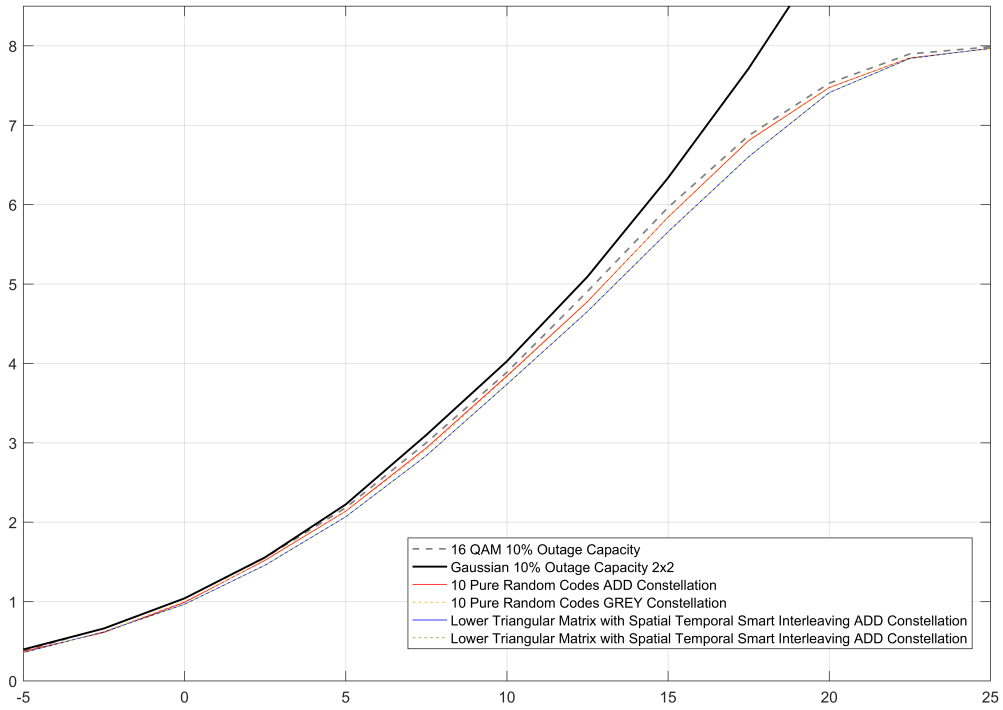

Figure 25: ADD and GREY Labelings used by 10 Pure Random Codes and lower triangular linear codes

3.6 Error Correcting Stage Tests

The purpose of this section is to simply check that a real world error-correcting stage works properly with this system. As seen in section 2.5.3, the stage selected for this test is the DVB-S2 LDPC long frame (64800 n-size) error correcting one.

Firstly, the proper rates for each (n, k) code to be used at every MLC level must be chosen. In order to do so, as explained in 3.2.4, we can use the chain rule to determine the rate needed at each bit/level. The testing environment of this test will be the following:

Test Configuration

MIMO Setup	2x2
Simulated Channels	100
Simulated Timeslots per Channel	64800
Constellation and labeling in use	16 QAM ADD
Demapper type	MAP
Error-correcting stage type	LDPC DVB-S2 Long-frame
Random Decoder Type	Factor Graph
Random Decoder Architecture	Architecture 3
Random Decoder Matrix Type	Lower triangular
Random Decoder Interleaver	Spatial Temporal Smart Interleaver

Reference plots used

Outage Capacity for 10 percent outage probability
16 QAM Capacity for 10 percent outage probability
10 Pure Random Codes with ADD labellings
10 Pure Random Codes with GREY labellings

With this environment, it becomes clear that the starting point for rate selection would be the rate per bit diagram that was already shown in 20. This graph shows, for every SNR level (x axis), which rates we would have to use for each MLC level (y axis). As these LDPC codes support r=1/4, 1/3, 2/5, 1/2, 3/5, 2/3, 3/4, 4/5, 5/6, 8/9, and 9/10, the resulting rates used per bit at each SNR level, taking into account a minimum 6 percent gap between the optimum rate and the realistic LDPC rate, would be the following ones:

	10 dB	12.5 dB	15 dB	17.5 dB	20 dB	22.5 dB	25 dB
Bit 1	0.2500	0.3333	0.4000	0.6667	0.8000	0.8889	0.9000
Bit 2	0.2500	0.4000	0.5000	0.6667	0.8333	0.9000	0.9000
Bit 3	0.3333	0.4000	0.6000	0.6667	0.8333	0.9000	0.9000
Bit 4	0.4000	0.5000	0.6000	0.7500	0.8333	0.9000	0.9000
Bit 5	0.4000	0.5000	0.6667	0.7500	0.8333	0.9000	0.9000
Bit 6	0.4000	0.6000	0.6667	0.8000	0.8333	0.9000	0.9000
Bit 7	0.5000	0.6000	0.6667	0.8000	0.8333	0.9000	0.9000
Bit 8	0.5000	0.6667	0.7500	0.8000	0.8333	0.9000	0.9000

The figure 26 shows the results that were produced by a DVB-S2 LDPC with such rates. This figure starts at 10dB unlike the previous ones, as DVB-S2 minimum rate is 1/4; smaller SNRs would need smaller rates than possible. In real life, in any case, one possibility in such situations would be to use smaller constellations at this point.

The results shown in the picture prove that a real error-correcting stage works without significant problems in conjunction with the novel random encoding and decoding stages.

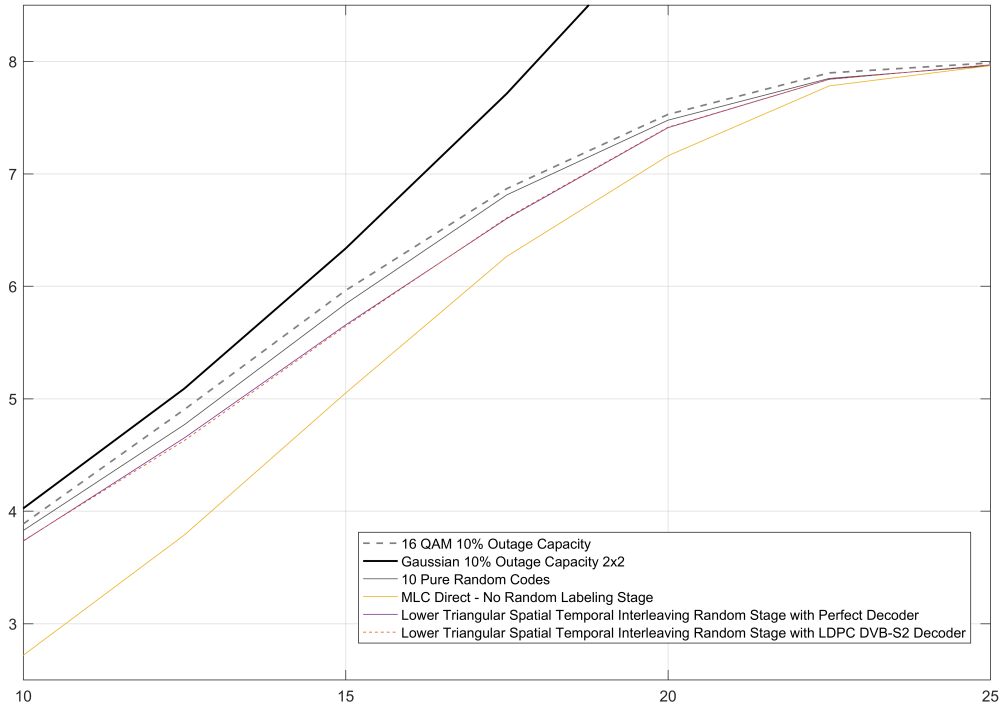

Figure 26: Lower triangular codes with LDPC DVB-S2 error correcting stage

4 Final Proposed System

4.1 Introduction

The main objective of this chapter is to briefly describe the components of the system that is proposed in this thesis. This system is merely intended to be the best one possible after all of the research and development done in this thesis, and all of the tests executed, the most important of which where show in chapter 1. The components chosen for the system are the ones that provide either the best tradeoff between spectral efficiency and computational complexity.

Each of the selected stages are briefly summarized. A full description of them is done in chapter 3.2.1; a reference is placed of the exact section in which the reader can find the full description of the stage.

4.2 Transmitter Stages

The final transmitter diagram 27 follows, as would be expected, the same general architecture than the one described in 2.1.2.

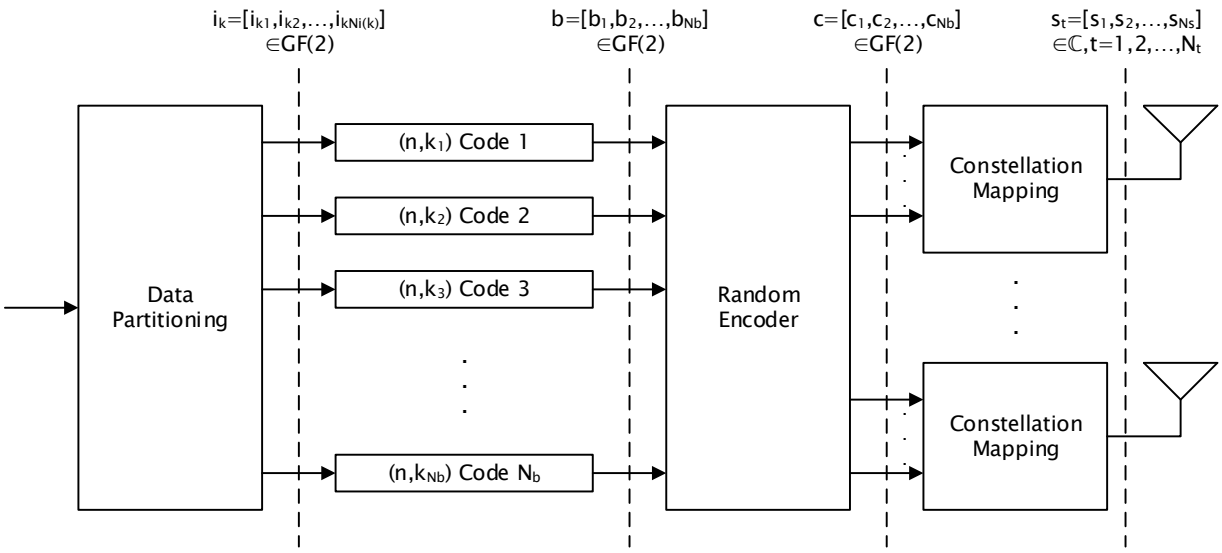

Figure 27: Transmitter Schematic

As mentioned in chapter 2.1, the transmitter input is expected to be a stream of bits **d** whose $P(d_i = 1) = P(d_i = 0) = 1/2$, in other words, that the stream has been compressed. Following this the stages that will be used by the proposed system will be the following:

- **Data Partitioning Stage**: As in MLC architectures in order to transmit N_b bits we use N_b error-correcting codes with different rates $(r_i, i = 1, 2, \ldots, N_b)$, the original

transmitted stream of bits **d** must be divided into N_b unevenly sized streams $\mathbf{k}_i$. The size of the streams will depend on the rate used for each i level. This stage is described with more detail in 2.1.2.

- **DVB-S2 LPDC Error-Correcting Encoding Stage**: A reliable and proven stage for the error-correcting codes is suggested, such as the DVB-S2 LDPC implemented in the software of this thesis. In any case, any Error-Correcting Stage that produces optimal results could be used at this stage. Given the nature of MLC systems, there will be N_b different encoders, each one with it's own different rate. Each $i-th$ encoder will receive the $\mathbf{k}_i$ vector and will output N_b vectors $\mathbf{n}_i$, all of which have the same size. In 2.5 discusses about the properties of this stage.

- **Random Encoder Stage**: Each $\mathbf{n}_i^T$ vector output by the previous stage could be considered a row of the matrix **N**. Each column of this matrix, of size $1 \times N_b$, would be the bits transmitted at the same time in the system, and also the **b** vector that this stage receives as input at each instant. In order to provide randomness to the sequence, as well as avoid intra-stage and inter-stage cycles (see 2.4.2), this stage will be composed of two substages:

  1. Linear Encoder: As demonstrated in , the best performing stage is the linear encoder, which converts the **b** bits and converts them to the coded **c** binary vectors by using its generator matrix **G**, **c**=**Gb**.
  2. Interleaver: The best performing interleaver (overall, for any generator matrix type) is, according to the demonstrated results of this thesis, the Spatial intra-block temporal smart interleaver. All **c** vectors in a error-correcting datablock are output by the Linear Encoder substage. Assuming that all of these **c** vectors could be stacked as column vectors of a matrix **C**, this stage performs random permutations both in row position and column position between all of it's elements.

  Both stages are described in detail in 2.4.4

- **Mapping Stage**: As we need an algorithmic constellation and labeling that can be quickly escalated to an arbitrary amount of bits, an ADD constellation labeling scheme such as the one described in 2.3.3 is used to convert all the resulting **c** vectors into electromagnetic signals with magnitude and phase. Each **c** vector sent from the previous stage is split into N_t (the number of transmit antennas) equally sized vectors, which are then sent to the antennas for their mapping into an ADD labeling and posterior transmission.

4.3 Receiver Stages

In the case of the proposed receiver diagram 28, it also follows, as would be expected, the same general architecture than the one described in 2.1.3.

The stages that will be used by the proposed system will be the following:

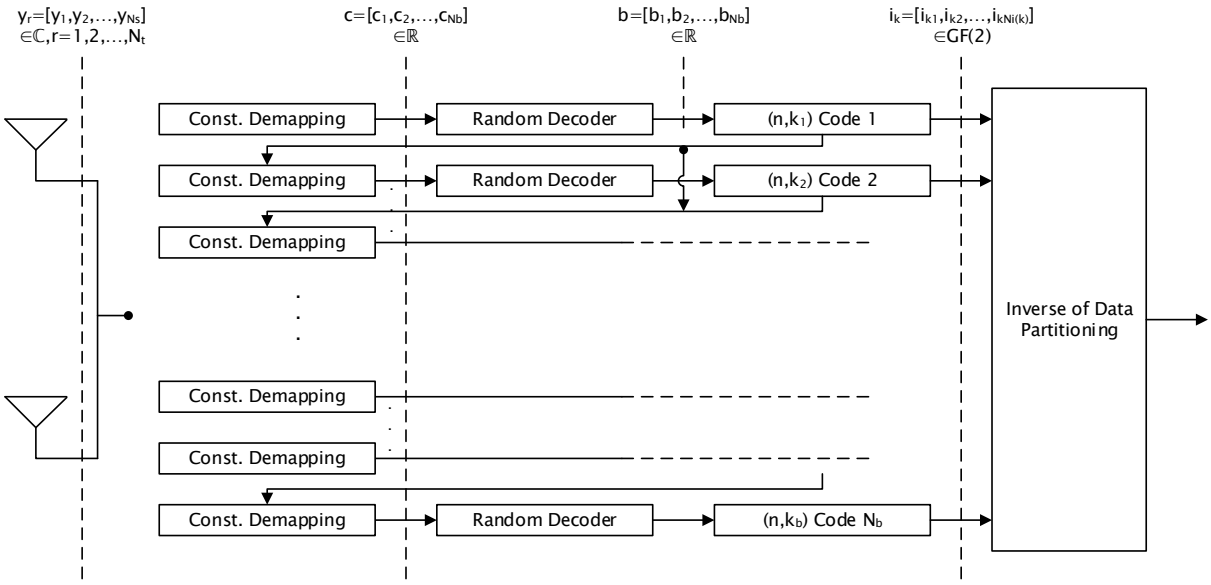

Figure 28: Receiver Schematic

- **Demapping Stage**: The signals are received at the N_r antennas, which create a $1 \times N_r$ sized $\mathbf{y}$ vector. This vector is demapped, taking into account that we're using ADD constellation and labeling methods such as the one described in 2.3.3, by either a MAP demapper (2.3.4), in the case one wants the maximum spectrum efficiency, a Binary MMSE demapper like the one developed for this thesis (2.3.5), in the case the simplest computational complexity is wanted or the number of bits to be transmitted is to high, or any other demapper if needed. The demapper creates a $\mathbf{c}$ vector of log likelihood ratios at every instant.

- **Random Decoder Stage**: This stage will be composed of two substages:

  1. Inverse interleaver: The inverse of the Spatial intra-block temporal smart interleaver done at the encoding phase is applied. All $\mathbf{c}$ vectors received from the demapper are stacked as if they were columns of a $\mathbf{C}$ matrix; the elements of this matrix are rearranged in order to return the c_{ij} elements of it to their original positions.
  2. Linear Decoder: The decoder receives the rearranged $\mathbf{c}$ vectors from the interleaver, and using a factorgraph representation of the equation $\mathbf{b}$=$\mathbf{Dc}$ recovers in terms of llrs, at the MLC level i, the $\mathbf{n}_i$ vector (a vector that contains all of the bits of a given MLC level).

  Both stages are described in detail in 2.4.4.

- **DVB-S2 LPDC Error-Correcting Decoding Stage**: A factor graph inverse of the error correcting code used in the encoder. At each MLC level i, it receives the $\mathbf{n}_i$ vector in terms of llrs and creates the i-th $\mathbf{k}_i$ binary vector, the output of the stage. On the other hand, if $i < N_b$ (the present MLC level is not the last one), the llr information generated at this stage is sent backwards to the linear decoder, interleaver

and demapper, and the process is started again at the demapper level for the $i+1$ level, with the *a priori* information found until now.

- **Data Partitioning Inverse Stage**: All N_b $\mathbf{k}_i$ vectors are joined again to create, if the reception has been successful, the original $\mathbf{d}$ vector.

Note that the flow of messages between the stages is described with more detail in 2.1.3.

5 Software Implementation

This chapter intends to explain how to operate the MIMO simulation environment created for testing the developed system.

The compressed file that contains the devised software has been arranged in two folders:

1. Simulator: It's composed of two important scripts:

1. Crear_Configuracio_Sim: It creates a config file needed to simulate the environment. It must be executed in order to create this file. The parameters that should be set are the following:

   - ConfigFileName: The name of the configuration file to be created.
   - nBitsAntenna: Number of bits sent by each antenna.
   - A_tx: Number of transmit antennas.
   - A_rx: Number of reception antennas.
   - nChannels: Number of different MIMO quasistatic fading channels, **H**, to be simulated for each SNR level.
   - nIter: Number of full MLC iterations to be executed (by default should be 1).
   - rEsNo: Vector that contains the range of SNRs, in dBs, to be tested at each simulation.
   - nSlots: Number of MIMO symbols transmitted for each channel (for each error-correcting datablock for quasistatic channels). If LDPC decoder is used, the size must be 64800.
   - Lineal_Code_Type: 'LinealCode' indicates that a linear random encoder/decoder is used by the system, 'Direct' means that no random encoder/decoder is used by the system, 'LinealLabel' uses a lock-up table to directly convert the received signals into uncoded **b** vectors, as in Lamarca's [1] paper.
   - RateFile: File which contains the base rates for the LDPC decoder. Typically this file will be the outage transmission rates calculated for the system by using a "Perfect Decoder" (see 2.5).
   - Lineal_Code.File: Name of the file that contains the **G** and **D** matrices of the lineal random code.
   - Interleaving_Mode:'SpatialTemporalSmartInterleaving', 'SpatialTemporalFullInterleaving', 'SpatialInterleaving', 'TemporalInterleaving' or 'NoInterleaving' for the different possible types of interleaver.
   - Demapper: 'MAP' or 'MMSE'.
   - Demapper_MMSE_Type: 'Apriori_ci' if the final Binary MMSE demapper is to be used, 'Apriori_bi' if the variant MMSE demapper described in chapter 2 is to be used.

- Decoder='PerfectDecoder' if the perfect decoder needs to be used, or 'LDPC-Scrambler' if it's the LDPC the one we want to use;

This script will create the configuration file in the Config subfolder.

2. **Simulacio_Auto**: This is the script that executes the simulations as defined in the config file generated in the previous step, and which have been moved to the folder *PendingTasks*. Once every config file simulation is finished, a results file will be stored in the folder *Results*, and the configuration file will be moved to *CompletedTasks*; the simulator will then execute any other config file present in *PendingTasks*. The results file will contain both the configuration of the simulation session, and a 'pseudo-mutual information' calculated for each bit in each channel and SNR level. Note that for this 'pseudo-mutual information' to be the real mutual information, the operation 1-'pseudo-mutual information' must be performed. In this thesis software, this is done by other software scripts such as the outage calculator.

2. Outage Calculator: The results from Simulacio_Auto should be copied here and, by executing main_calcul_outage_automode the outage transmission rates of any results file present in the folder (which have a name starting with IM_) are calculated. By default, the outage is 10 percent, but this and the number of bits used by the MIMO system can be changed in the script main_calcul_outage_automode.

3. Result Visualization: A series of scripts that where used to visualize the capacity, capacity per bit and dispersion plots. Among many, some of the most important would be Comparar_Grafiques _Capacitat, Comparar_Grafiques _CapacitatxBit and Comparar_Grafiques _Dispersio, which show the plots described at the beginning of this description.

4. Discarded Designs Software: This is merely a repository of the large amount of scripts that were programed in order to do the research and development of this thesis. They include the simulators for the architectures 1 and 2 of the final proposed system (which was discarded for the architecture 3). Other scripts of this repository were used for the simulation of the non-linear systems that were devised at first as the solutions for this study.

6 Conclusion

The results obtained from the simulation of the system devised in this thesis demonstrates that a very simple stage, the linear random encoding/decoding, can minimize outage related losses without suboptimal initialization phases and introducing delays as in the case of D-BLAST architectures, nor requiring iteration between the demapper and the decoder stages such as the case of SpaceTime Codes. All the operations required by this stage are carried in a single error-correcting datablock, keeping the delays at the minimum level.

The research carried in order to reach these results has been long and complex; as shown in the second chapter, many random stages were devised and all of them had some, at the time, insurmountable problems that required modifications. As seen in 2.4.3, Standard Random Generators, Original Lineal Codes, Selected Lineal Codes, Original Non-Lineal Codes, Stacked Non-Lineal Codes, Pseudotriangular and Recursive Non-Lineal Codes with Interleaving and Pseudotriangular and Recursive Lineal-Codes With Interleaving were evaluated before reaching the final simpler proposal. At one point the thesis was already being documented for the last two designs, until finally it was discovered that with the proper linear stages, which were much simpler in nature, a optimal design was also possible.

The proposed linear random encoder and decoder meets positively the objectives set for it:

- Both stages are computationally very simple, with computational complexity proportional to the number of bits (N_b) instead of exponential (2_b^N), as was the case of the lock-up tables used in [1].
- Both stages are very easy to design for a trivial number of bits. Both use as generator and decoding matrices ($\mathbf{G}$ and $\mathbf{D}$) lower triangular sparse matrices. The tests carried in this thesis' research showed that any sparse matrix without short-length cycles operates optimally.
- The decoding stage works with llrs, and is able to send output llrs to the error-correcting stage for it's optimal processing.
- The system is able to get spectral efficiencies near the outage capacity.

Furthermore, an additional non-intended advantage was found with this system: it's required rates per bit of the first MLC levels are much higher than in the original lock-up table random encoder system ([1], and also in Results section 3.3.6 the rates per bit can be seen). This is much easier to accomplish with current block error correcting codes than the rates that were close to zero that were needed in the second case's first MLC levels.

Figures 29, 30, 31 are the proof that the devised linear random encoder and decoder fulfills its objectives. The first figure demonstrates its spectral efficiency against the [1] lock-up table method and the outage capacity limits (both constellation size constrained and gaussian). The second demonstrates the advantage mentioned in the previous paragraph, and the third one compares the channels dispersion between the [1] random encoding system and the final

proposed system of the thesis. As explained in the results section, the second one, while having slightly higher dispersion in the first and last levels, tend to form a straight line rather than a curve, a fact that explains why the rates per bit have the advantageous disposition mentioned in the previous paragraph.

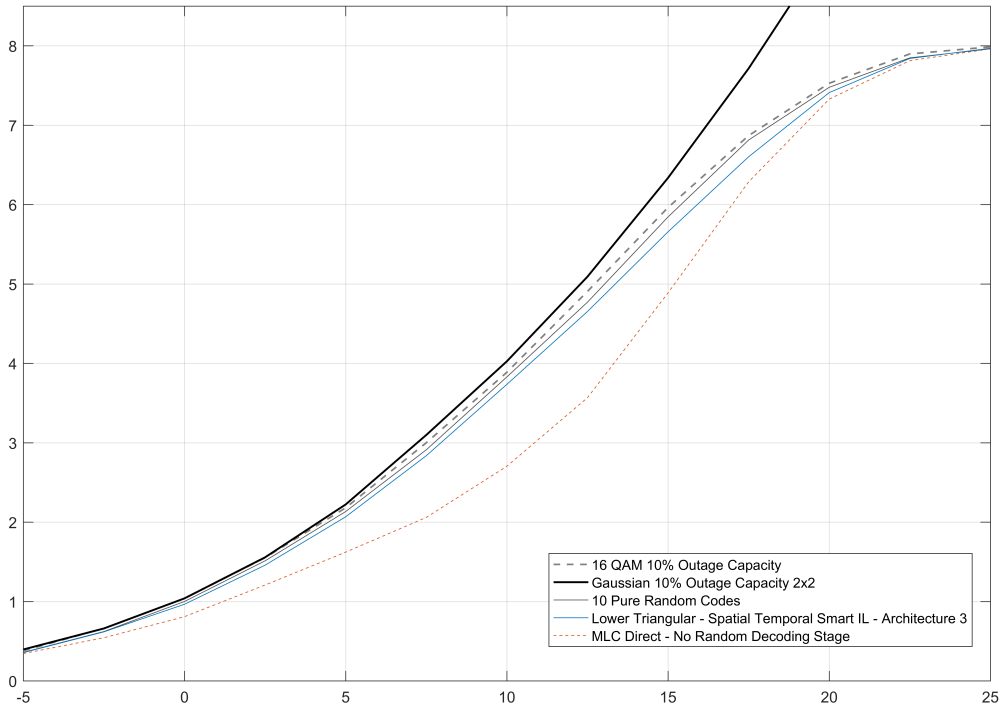

Figure 29: 10 percent Outage Capacity for the Proposed System (with lower triangular matrix and spatial temporal interleaver)

On the other hand, the results of the Binary MMSE demapper devised for this thesis are not so brilliant. The spectral performance without any demapper-decoder iteration has room for improvements. There is a lot of future potential in this point, as several papers demonstrate that MMSE-like demapper behave very well once iterations are allowed between these stages [12]. Despite this, the demapper as of now fulfills the following very interesting features:

- Both stages are computationally very simple, allowing execution in non-exponential time for a large number of bits per MIMO channel use. The complexity of the stage as of now is proportional to $N_b \times N_b$, much smaller than exponential requirements, and has plenty of room for optimizations.
- Both stages are very easy to design for a trivial amount of bits and antennas.
- The demapping stage is able to feed *soft* output values to the posterior stages.

The main field of research on this subject would have to focus on iterative schemes between the MMSE demapper and the rest of the stages. This is something, unfortunately, that is

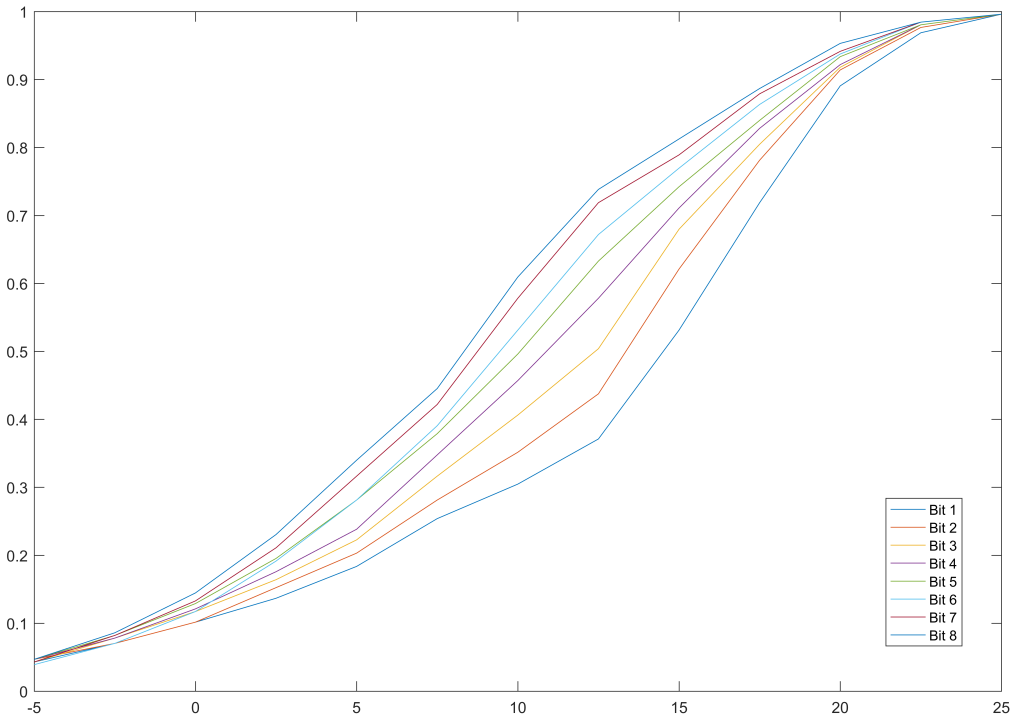

Figure 30: Bit Rate for Lower Triangular with Spatial Temporal Smart Interleaving in architecture 3

beyond of the scope of the present thesis. Afterall this thesis is a testament to the fact that persistence pays-off, but also that the lack of boundaries and priorities may lead to a very protracted developement phase.

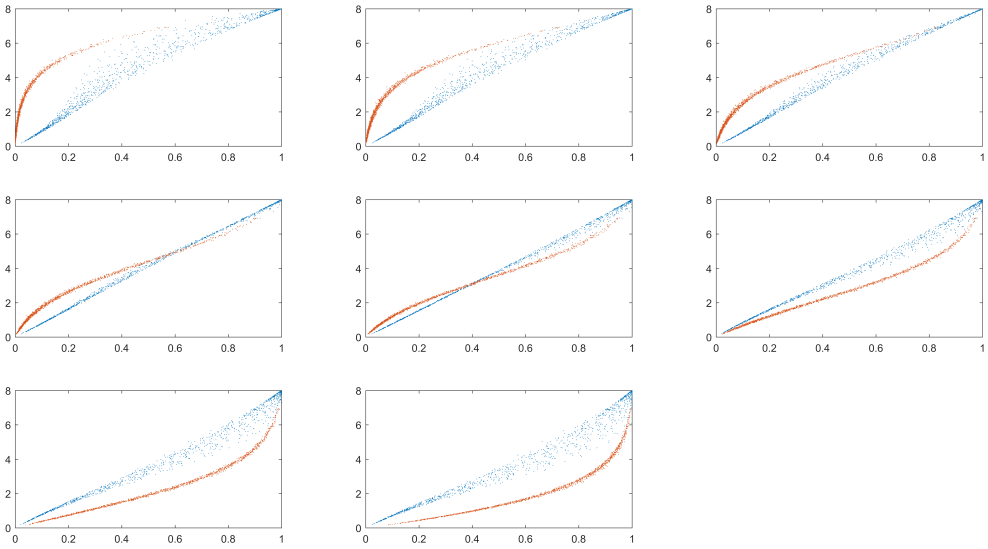

Figure 31: Transmission rate per bit vs Total channel transmission rate dispersion. In blue, triangular lineal codes, in orange, 10 pure random codes

7 Annex

The annex in this thesis includes the most important information of the documentation that at one point was prepared to be presented as the proposed solution of the thesis. As already mentioned in chapter 2, these schemes were discarded when it was found that a much simpler approach was possible. Nevertheless, we attach this techniques here because of the following reasons:

1. They illustrate novel ideas that, for the decoding stage, use both the bits received from the demapper *and* those already decoded.

2. In the case of the non-linear random encoder and decoder, provide a way to generate invertible non-linear codes very quickly and efficiently.

3. They show part of the big effort that was carried to develop these solutions, even if finally they were not needed in this case

The notation in this part is slightly different than in the rest of the thesis: **vectors in this annex are be defined as row vectors rather than column vectors.**

7.1 Pseudotriangular Recursive Linear Encoder

Let $\mathbf{b} = [b_1\ b_2\ b_3\ \dots\ b_{N_b}]$ be the N_b input bits of the coder, while the vector $\mathbf{c} = [c_1\ c_2\ c_3\ \dots\ c_{N_b}]$ represents the coded output bits of the stage. We define the compound vector $[\mathbf{c}\ \mathbf{b}] = [c_1\ c_2\ c_3\ \dots\ c_{N_b}\ b_1\ b_2\ b_3\ \dots\ b_{N_b}]$ and the $N_b \times (2.N_b)$ matrix $\mathbf{G}$ such that:

$$\mathbf{G} = \begin{bmatrix} 0 & \cdots & 0 & 0 & g_{11} & \cdots & g_{1(N_b-2)} & g_{1(N_b-1)} & 1 \\ 0 & \cdots & 0 & g_{21} & g_{22} & \cdots & g_{2(N_b-1)} & 1 & 0 \\ 0 & \cdots & g_{31} & g_{32} & g_{33} & \cdots & 1 & 0 & 0 \\ \vdots & \ddots & \vdots & \vdots & \vdots & \ddots & \vdots & \vdots & \vdots \\ 0 & \cdots & g_{N_b(N_b-2)} & g_{N_b(N_b-1)} & 1 & \cdots & 0 & 0 & 0 \end{bmatrix} \tag{76}$$

Where $\mathbf{G}$ defines which input and output bits will be used for each output bit. Given these vectors and matrix the system's output can be calculated from the following expressions:

$$\begin{bmatrix} 0 & \cdots & 0 & 1 \\ 0 & \cdots & 1 & 0 \\ \vdots & \ddots & \vdots & \vdots \\ 1 & \cdots & 0 & 0 \end{bmatrix} \mathbf{c}^T = \mathbf{G}[\mathbf{c}\ \mathbf{b}]^T \tag{77}$$

$$\begin{bmatrix} c_{N_b} \\ c_{N_b-1} \\ c_{N_b-2} \\ \vdots \\ c_1 \end{bmatrix} = \begin{cases} b_1 g_{11} + b_2 g_{12} + \cdots + b_{N_b-1} g_{1(N_b-1)} + b_{N_b} \\ c_{N_b} g_{21} + b_1 g_{22} + \cdots + b_{N_b-2} g_{2(N_b-1)} + b_{N_b-1} \\ c_{N_b-1} g_{31} + c_{N_b} g_{32} + \cdots + b_{N_b-3} g_{3(N_b-1)} + b_{N_b-2} \\ \vdots \quad\quad \vdots \quad\quad \ddots \quad\quad \vdots \quad\quad \vdots \\ c_2 g_{N_b 1} + c_3 g_{N_b 2} + \cdots + c_{N_b-1} g_{N_b(N_b-1)} + b_1 \end{cases} \tag{78}$$

Note how c_{N_b} is an independent term which depends entirely on known matrix $\mathbf{G}$ and vector $\mathbf{b}$. Each element $c_i, i < N_b$ depends, at most, on the results of already calculated elements $[c_{i+1}\ c_{i+2}\ \cdots\ c_{N_b}]$ besides $\mathbf{G}$ and $\mathbf{b}$. This allows a straightforward, iterative codification of each output.

Interestingly, the codes resulting from the expressions (78) can be separated in two groups:

- If $g_{ij} = g_j\ \forall i$, the code is actually a pseudo-LFSR (*Linear Feedback Shift Register*). In this case, as illustrated in the example figure (32), the structure of the code remains constant through the process, while at the same time past outputs of the code are used to calculate new outputs, like a LFSR. However, typical LFSR constraints, such as the requirement of finding a maximal length sequence code, doesn't apply in this case; indeed, performance of the code will be influenced by other factors.
- If $g_{ij} \neq g_j\ \forall i$, the code structure varies for each output bit.

It's worth noticing that the algorithm presented in this section produces a reversible code for any $\mathbf{G}$ matrix structured as shown in the expression (76).

**Proof:* The most common method of generation of linear codes is through the use of the Euclidean inner product between the N_b-bit binary input $\mathbf{b}$ and each of the columns or rows of the $N_b \times N_b$ generator matrix of the code $\mathbf{G}_s$:

$$\mathbf{c} = \mathbf{b}\mathbf{G}_s \leftrightarrow \mathbf{c}^T = \mathbf{G}_s \mathbf{b}^T \tag{79}$$

The code $\mathbf{c}$ generated is invertible if and only if $\text{rank}\,(\mathbf{G}_s) = N_b$. Consequently, if for any possible $N_b \times 2.N_b$ $\mathbf{G}$ matrix it's possible to find an equivalent standard linear code generator matrix, $\mathbf{G}_s$, whose rank is full, the hypothesis will be proven. Every coded bit $c_i, 1 \leq i \leq N_b$ in the system of equations (78) can be expressed entirely as a set of *ands* and *xors* between $\mathbf{b}$ and $\mathbf{G}$ elements, by replacing every variable $c_{i+j}, i + j \leq N_b$ by its previously calculated value. The following expression produces a result identical to the solution of (78) in terms of $\mathbf{b}$ and $\mathbf{G}$ elements:

$$\mathbf{P} \underbrace{\begin{bmatrix} c_1 \\ c_2 \\ c_3 \\ \vdots \\ c_{N_b} \end{bmatrix}}_{\mathbf{c}^T} = \underbrace{\begin{bmatrix} 1 & 0 & \cdots & 0 & 0 \\ g_{21} & 1 & \cdots & 0 & 0 \\ g_{32} & g_{31} & \cdots & 0 & 0 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ g_{N_b(N_b-1)} & g_{N_b(N_b-2)} & \cdots & g_{N_b 1} & 1 \end{bmatrix}}_{\mathbf{F}_1} \underbrace{\begin{bmatrix} g_{11} & \cdots & g_{1(N_b-2)} & g_{1(N_b-1)} & 1 \\ g_{22} & \cdots & g_{2(N_b-1)} & 1 & 0 \\ g_{33} & \cdots & 1 & 0 & 0 \\ \vdots & \ddots & \vdots & \vdots & \vdots \\ 1 & \cdots & 0 & 0 & 0 \end{bmatrix}}_{\mathbf{F}_2} \underbrace{\begin{bmatrix} b_1 \\ b_2 \\ b_3 \\ \vdots \\ b_{N_b} \end{bmatrix}}_{\mathbf{b}^T}$$

Where the permutation matrix P is antidiagonal such that $\{p_{ij} = 1 \backslash i + j = N_b + 1\}$. Equivalently:

$$\begin{aligned} \mathbf{P}\mathbf{c}^T &= \mathbf{F}_1\mathbf{F}_2\mathbf{b}^T \\ \mathbf{P}\mathbf{P}\mathbf{c}^T &= \mathbf{P}\mathbf{F}_1\mathbf{F}_2\mathbf{b}^T \\ \mathbf{I}\mathbf{c}^T &= \mathbf{P}\mathbf{F}_1\mathbf{F}_2\mathbf{b}^T \\ \mathbf{c}^T &= \mathbf{P}\mathbf{F}_1\mathbf{F}_2\mathbf{b}^T \end{aligned} \tag{80}$$

Using the expressions in (79) and (80) the following equivalence becomes clear:

$$\mathbf{P}\mathbf{F}_1\mathbf{F}_2 = \mathbf{G}_s$$

Given that both $\mathbf{F}_1$ and $\mathbf{F}_2$ are $N_b \times N_b$ square matrices we can use the determinants to verify if $\mathbf{G}_s$ is a full rank matrix. The rank of a matrix $\mathbf{A}$ is full if and only if $\det(\mathbf{A}) \neq 0$. On the other hand, if $\mathbf{C} = \mathbf{A}\mathbf{B}$ then $\det(\mathbf{C}) = \det(\mathbf{A})\det(\mathbf{B})$. Hence $\det(\mathbf{G}_s) = \det(\mathbf{P})\det(\mathbf{F}_1)\det(\mathbf{F}_2)$. While obtaining the determinant of an arbitrarily large matrix is usually a time consuming task, a particular property of matrices like $\mathbf{P}$, $\mathbf{F}_1$ and $\mathbf{F}_2$ can be exploited. The determinant of a lower or upper triangular matrix such a $\mathbf{F}_1$ is simply the product of its diagonal elements: $\mathbf{A} : \{a_{ij} = 0 \backslash i > j\}$ or $\mathbf{A} : \{a_{ij} = 0 \backslash i < j\}$ then $\det(\mathbf{A}) = \prod a_{ii}$. Neither $\mathbf{P}$ nor $\mathbf{F}_2$ are triangular matrices, but they can be easily converted to one by gaussian elimination in the form of row permutations, taking into consideration that each permutation means a change in the determinant sign. Flipping $\mathbf{P}$ or $\mathbf{F}_2$ along the horizontal axis transform them into an identity matrix and a lower triangular matrix, respectively, and costs $\lfloor N_b/2 \rfloor$ row permutations each. Finally:

$$\begin{aligned} \det(\mathbf{G}_s) &= \det(\mathbf{P})\det(\mathbf{F}_1)\det(\mathbf{F}_2) \\ &= (-1)^{\lfloor N_b/2 \rfloor}.1.(-1)^{\lfloor N_b/2 \rfloor} = 1 \neq 0 \rightarrow \mathbf{G}_s \text{ is a full rank matrix } \forall \mathbf{G} \end{aligned} \tag{81}$$

Hence proving that every possible $\mathbf{G}$ matrix structured as shown in (76) generates a full, invertible linear code.

Example: Let $\mathbf{b} = [1\ 1\ 1\ 1\ 1\ 1\ 1\ 0]$ and $\mathbf{G}$:

$$\mathbf{G} = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

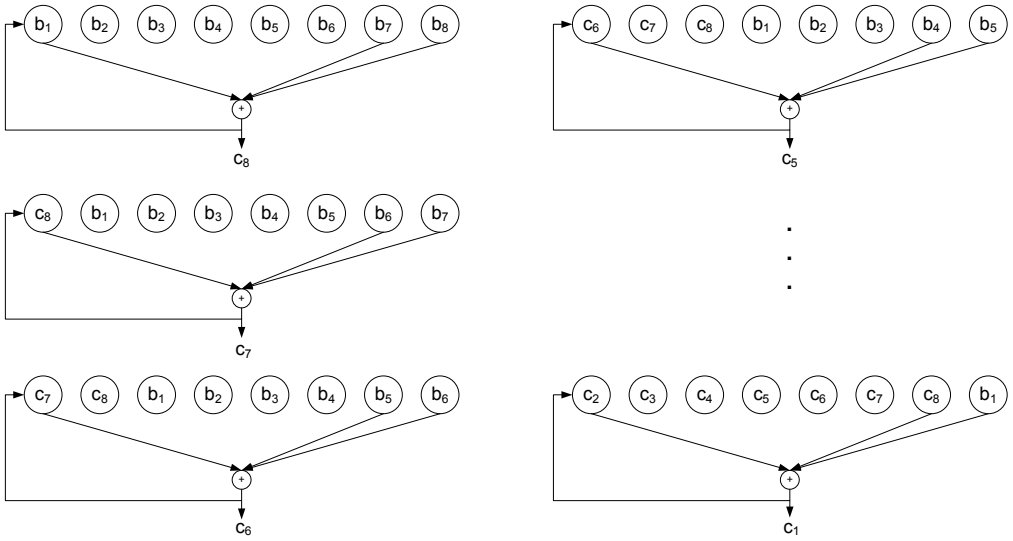

Figure 32: LFSR-like diagram of the encoding scheme

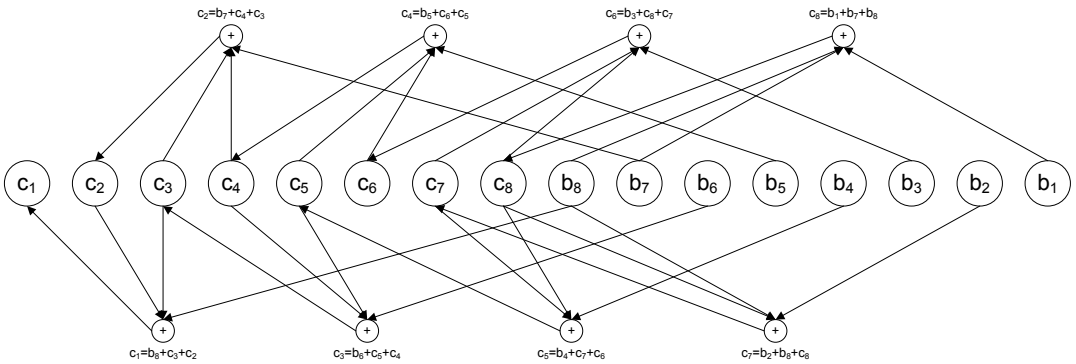

Figure 33: Factor-graph style diagram of the encoding process

This particular **G** matrix belongs to a subset of possible generator matrices that bear close resemblance to LSFRs. Given that $g_{ij} = g_j \ \forall i$, the structure of the code remains constant at every iteration and the only differences between output bits arise from the binary shifts and feedbacks, mimicking those found on such linear codes. Figure (32) illustrates this concept. In any case, either this or other types of generator matrices are equally simple to use as coding devices. Each output bit, c_i is computed iteratively, using the result of the last calculated elements, c_{i-n} $\{n \in \mathbb{N} \backslash n < i\}$, as feedback. The last output bit is the first one to

be encoded as it has no dependency upon any previous coded element. In this case:

$$\begin{aligned}
c_8 &= b_1g_{11} + b_2g_{12} + b_3g_{13} + b_4g_{14} + b_5g_{15} + b_6g_{16} + b_7g_{17} + b_8 \\
&= 1.1 + 1.0 + 1.0 + 1.0 + 1.0 + 1.0 + 1.1 + 0 = 0 \\
c_7 &= c_8g_{21} + b_1g_{22} + b_2g_{23} + b_3g_{24} + b_4g_{25} + b_5g_{26} + b_6g_{27} + b_7 \\
&= 0.1 + 1.0 + 1.0 + 1.0 + 1.0 + 1.0 + 1.1 + 1 = 0 \\
c_6 &= 0.1 + 0.0 + 1.0 + 1.0 + 1.0 + 1.0 + 1.1 + 1 = 0 \\
c_5 &= 0.1 + 0.0 + 0.0 + 1.0 + 1.0 + 1.0 + 1.1 + 1 = 0 \\
c_4 &= 0.1 + 0.0 + 0.0 + 0.0 + 1.0 + 1.0 + 1.1 + 1 = 0 \\
c_3 &= 0.1 + 0.0 + 0.0 + 0.0 + 0.0 + 1.0 + 1.1 + 1 = 0 \\
c_2 &= 0.1 + 0.0 + 0.0 + 0.0 + 0.0 + 0.0 + 1.1 + 1 = 0 \\
c_1 &= 0.1 + 0.0 + 0.0 + 0.0 + 0.0 + 0.0 + 0.1 + 1 = 1
\end{aligned}$$

In practice, using this fairly limited set of binary *and* and *xor* operations the output vector $\mathbf{c} = [1\ 0\ 0\ 0\ 0\ 0\ 0\ 0]$ is obtained. The figure (33) illustrates graphically this process (with the encoding order being from right to left), with a factor-graph style that's useful for drawing parallels with the decoder.

7.2 Pseudotriangular Recursive Non-Linear Encoder

7.2.1 Introduction

Let D be the degree of the non-linear code. Given $\mathbf{b} = [b_1\ b_2\ b_3\ \ldots\ b_{N_b}]$, the $1 \times N_b$ vector containing the input bits, and $\mathbf{G}_s$, a $N_b \times N_b^D$ a *generic* D degree non-linear transformation can be expressed as:

$$\mathbf{c} = \underbrace{(\mathbf{b} \otimes \mathbf{b} \otimes \ldots \otimes \mathbf{b})}_{D-1\ \otimes\ \text{operations}} \mathbf{G}_s^T \tag{82}$$

Where $\mathbf{c} = [c_1\ c_2\ c_3\ \ldots\ c_{N_b}]$ is the $1 \times N_b$ output vector, and the operation '$\otimes$' is the Kronecker tensor product. For a $m \times n$ matrix $\mathbf{A}$ and a $p \times q$ matrix $\mathbf{B}$, the aforementioned Kronecker product is defined as:

$$\mathbf{A} \otimes \mathbf{B} \triangleq \begin{bmatrix} \mathbf{a}_{11}\mathbf{B} & \mathbf{a}_{12}\mathbf{B} & \cdots & \mathbf{a}_{1n}\mathbf{B} \\ \mathbf{a}_{21}\mathbf{B} & \mathbf{a}_{22}\mathbf{B} & \cdots & \mathbf{a}_{2n}\mathbf{B} \\ \vdots & \vdots & \ddots & \vdots \\ \mathbf{a}_{m1}\mathbf{B} & \mathbf{a}_{m2}\mathbf{B} & \cdots & \mathbf{a}_{mn}\mathbf{B} \end{bmatrix} \in GF(2)^{mp \times nq} \tag{83}$$

The $D-1$ kronecker tensor products done over $\mathbf{b}$ create a vector containing *all* the possible linear and non-linear terms of a non-linear expression of degree D. On the other hand, the

$N_b \times N_b^D$ matrix $\mathbf{G}$ is the *generator* matrix of the nonlinear code, a construct that defines which linear and non-linear terms of the kronecker tensor products will be used to calculate each output. Non-linearities are introduced by using binary ANDs during the calculation of each term of the kronecker tensor products, while a final operation to determine each output term is a bilinear binary XOR, which adds each linear or nonlinear term related to the output as determined by $\mathbf{G}_s$.

Note that the equation (82) produces a non-linear transformation that doesn't assure reversibility by itself; indeed, only a tiny subset of all the possible $\mathbf{G}_s$ matrices actually create a fully reversible code.

7.2.2 Proposed non-linear encoder

In order to have the possibility to create a coding scheme reversible for any given configuration, an specific arrangement of the generator matrix has to be defined. Let matrices $\mathbf{M}$ and $\mathbf{L}$, both $N_b \times 2N_b$ partitioned matrices be defined as:

$$\mathbf{M} = \left[\begin{array}{ccccccccccccc} 0 & 0 & 0 & \cdots & 0 & 0 & 1 & 1 & \cdots & 1 & 1 & 0 \\ \hline 0 & 0 & 0 & \cdots & 0 & 1 & 1 & 1 & \cdots & 1 & 0 & 0 \\ \hline 0 & 0 & 0 & \cdots & 1 & 1 & 1 & 1 & \cdots & 0 & 0 & 0 \\ \hline \vdots & \vdots & \vdots & \ddots & \vdots & \vdots & \vdots & \vdots & \ddots & \vdots & \vdots & \vdots \\ \hline 0 & 0 & 1 & \cdots & 1 & 1 & 1 & 0 & \cdots & 0 & 0 & 0 \\ \hline 0 & 1 & 1 & \cdots & 1 & 1 & 0 & 0 & \cdots & 0 & 0 & 0 \end{array}\right]^T$$

$$\mathbf{L} = \left[\begin{array}{ccccccccccccc} 0 & 0 & 0 & \cdots & 0 & 0 & 0 & 0 & \cdots & 0 & 0 & 1 \\ \hline 0 & 0 & 0 & \cdots & 0 & 0 & 0 & 0 & \cdots & 0 & 1 & 0 \\ \hline 0 & 0 & 0 & \cdots & 0 & 0 & 0 & 0 & \cdots & 1 & 0 & 0 \\ \hline \vdots & \vdots & \vdots & \ddots & \vdots & \vdots & \vdots & \vdots & \ddots & \vdots & \vdots & \vdots \\ \hline 0 & 0 & 0 & \cdots & 0 & 0 & 0 & 1 & \cdots & 0 & 0 & 0 \\ \hline 0 & 0 & 0 & \cdots & 0 & 0 & 1 & 0 & \cdots & 0 & 0 & 0 \end{array}\right]^T \tag{84}$$

The matrix $\mathbf{M}$ will serve as a basis for determining which non-linear operations can be done for each output bit without compromising both reversibility and fast calculation of the decoding matrix. On the other hand, the matrix $\mathbf{L}$ points exactly which linear operations will be mandatory in order to preserve reversibility. Given these defined matrices and vectors, the general non-linear encoder expression is:

$$\begin{bmatrix} 0 & \cdots & 0 & 1 \\ 0 & \cdots & 1 & 0 \\ \vdots & \ddots & \vdots & \vdots \\ 1 & \cdots & 0 & 0 \end{bmatrix} \mathbf{c} = \underbrace{([\mathbf{c}\ \mathbf{b}] \otimes [\mathbf{c}\ \mathbf{b}] \otimes \ldots \otimes [\mathbf{c}\ \mathbf{b}])}_{D-1\ \otimes\ \text{operations}} (\mathbf{G}^T \circ \underbrace{(\mathbf{M} \odot \mathbf{M} \odot \ldots \odot \mathbf{M})}_{D-1\ \odot\ \text{operations}} + \underbrace{(\mathbf{L} \odot \mathbf{L} \odot\ \ldots\ \odot \mathbf{L})}_{D-1\ \odot\ \text{operations}}) \tag{85}$$

Where the operation '$\odot$' is the Khatri-Rao column-wise product, defined as:

$$\begin{array}{c} \mathbf{A} \in GF(2)^{m \times n} \quad , \quad \mathbf{B} \in GF(2)^{p \times n} \\ \mathbf{A} = \left[\begin{array}{c|c|c|c} \mathbf{a}_1 & \mathbf{a}_2 & \ldots & \mathbf{a}_n \end{array}\right] \quad , \quad \mathbf{B} = \left[\begin{array}{c|c|c|c} \mathbf{b}_1 & \mathbf{b}_2 & \ldots & \mathbf{b}_n \end{array}\right] \end{array}$$

$$\mathbf{A} \odot \mathbf{B} \triangleq \left[\begin{array}{c|c|c|c} \mathbf{a}_1 \otimes \mathbf{b}_1 & \mathbf{a}_2 \otimes \mathbf{b}_2 & \cdots & \mathbf{a}_n \otimes \mathbf{b}_n \end{array} \right] \in GF(2)^{mp \times n} \tag{86}$$

While the operator '$\circ$' represents the Hadamard element-wise product of the partitioned matrices, which can be illustrated as:

$$\mathbf{A} \in GF(2)^{m \times n} \quad , \quad \mathbf{B} \in GF(2)^{m \times n}$$

$$\mathbf{A} \circ \mathbf{B} \triangleq \left[\begin{array}{c|c|c|c} a_{11}b_{11} & a_{12}b_{12} & \cdots & a_{1n}b_{1n} \\ \hline a_{21}b_{21} & a_{22}b_{22} & \cdots & a_{2n}b_{2n} \\ \hline \vdots & \vdots & \ddots & \vdots \\ \hline a_{m1}b_{m1} & a_{m2}b_{m2} & \cdots & a_{mn}b_{mn} \end{array} \right] \in GF\,(2)^{m \times n} \tag{87}$$

Finally, $\mathbf{G}$ is once again the $N_b \times (2N_b)^D$ generator matrix, defined as:

$$\mathbf{G} = \begin{bmatrix} \mathbf{0} & \cdots & \mathbf{0} & \mathbf{0} & \mathbf{g}_{11} & \cdots & \mathbf{g}_{1(N_b-2)} & \mathbf{g}_{1(N_b-1)} & \mathbf{0} \\ \mathbf{0} & \cdots & \mathbf{0} & \mathbf{g}_{21} & \mathbf{g}_{22} & \cdots & \mathbf{g}_{2(N_b-1)} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \cdots & \mathbf{g}_{31} & \mathbf{g}_{32} & \mathbf{g}_{33} & \cdots & \mathbf{0} & \mathbf{0} & \mathbf{0} \\ \vdots & \ddots & \vdots & \vdots & \vdots & \ddots & \vdots & \vdots & \vdots \\ \mathbf{0} & \cdots & \mathbf{g}_{N_b(N_b-2)} & \mathbf{g}_{N_b(N_b-1)} & \mathbf{0} & \cdots & \mathbf{0} & \mathbf{0} & \mathbf{0} \end{bmatrix} \tag{88}$$

Where $\mathbf{g}_{ij}$ are the $1 \times (2N_b)^{D-1}$ vectors containing the operations that will be done by the non-linear encoder, while the bold $\mathbf{0}$ are equally sized vectors whose elements are all 0. Note that this generator matrix is, in any case, further constrained by the matrix M, which will nullify many elements of $\mathbf{G}$ in order to preserve the encoding scheme. Given the equation (85) the encoder's main system of equations can be illustrated as:

$$\begin{bmatrix} c_{N_b} \\ c_{N_b-1} \\ c_{N_b-2} \\ \vdots \\ c_1 \end{bmatrix} = \begin{cases} b_1 g_{11}^{N_b+1} + b_1 b_2 g_{11}^{N_b+2} + \cdots + b_{N_b-1} b_{N_b-1} g_{1(N_b-1)}^{(2N_b)^{D-1}-1} + b_{N_b} \\ c_{N_b} g_{21}^{N_b} + c_{N_b} b_1 g_{21}^{N_b+1} + \cdots + b_{N_b-2} b_{N_b-2} g_{2(N_b-1)}^{(2N_b)^{D-1}-2} + b_{N_b-1} \\ c_{N_b-1} g_{31}^{N_b-1} + c_{N_b-1} c_{N_b} g_{31}^{N_b} + \cdots + b_{N_b-3} b_{N_b-3} g_{3(N_b-1)}^{(2N_b)^{D-1}-3} + b_{N_b-2} \\ \vdots \quad\quad\quad\quad \vdots \quad\quad\quad\quad \ddots \quad\quad\quad\quad \vdots \quad\quad\quad\quad \vdots \\ c_2 g_{N_b 1}^2 + c_2 c_3 g_{N_b 1}^3 + \cdots + c_{N_b} c_{N_b} g_{N_b(N_b-1)}^{(2N_b)^{D-1}-N_b} + b_1 \end{cases} \tag{89}$$

The notation g_{ij}^k is entirely equivalent to the more orthodox $g_{ij}(k)$: g_{ij}^k represents the k-th element of the $\mathbf{g}_{ij}$ vector. On the other hand, some terms of the system have been simplified. For instance, each $a_i a_i$ element has been simplified to a_i, as this equivalence holds true in a binary system. The first and last columns of the equation (89) have been simplified by this procedure. Moreover, the effects of the $\mathbf{M}$ and $\mathbf{L}$ matrices have been taken into consideration, albeit not explicitly showing them as they're mere constants. The last column of the system is the linear term introduced by $\mathbf{L}$, while no g_{ij}^k with $k = (2N_b)^{D-1}$ is shown as they're always nullified by $\mathbf{M}$.

Note that in the expression (89), as in the linear case shown in (78), only the c_{i-n}, $n < i$ terms already calculated or the known elements of $\mathbf{b}$ are used to determine c_i, hence the system is easily solved if the equations are processed sequentially.

As in the case of the linear codes, the proposed non-linear encoder produces codes that can be separated in two categories:

- If $\mathbf{g}_{ij} = \mathbf{g}_j \ \forall i$, the code is a pseudo-NLFSR (*Non Linear Feedback Shift Register*). Although NLFSR behaviour isn't completely understood in terms of finding the maximal length sequence, this constraint isn't important for the scope of this family of encoders; reversibility and randomness, are the only requirements for these encoders.
- If $\mathbf{g}_{ij} \neq \mathbf{g}_j \ \forall i$, the code structure varies for each output bit.

Although it hasn't been backed by a mathematical proof as in the linear case, experimental testing seem to indicate that the codes generated with this method are always fully reversible.

Example: Let a random encoder be a second degree system with three input bits and a unspecified matrix $\mathbf{G}$. Hence $N_b = 3$ and $D = 2$. The system's matrices will be defined as:

$$\mathbf{M} = \left[\begin{array}{c|c|c} 0 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & 1 & 1 \\ 1 & 1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 0 \end{array}\right] \quad \mathbf{L} = \left[\begin{array}{c|c|c} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & 1 & 0 \\ 1 & 0 & 0 \end{array}\right] \quad \mathbf{G}^T = \begin{bmatrix} \mathbf{0} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{g}_{31} \\ \mathbf{0} & \mathbf{g}_{21} & \mathbf{g}_{32} \\ \mathbf{g}_{11} & \mathbf{g}_{22} & \mathbf{0} \\ \mathbf{g}_{12} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{0} \end{bmatrix}$$

The encoding equation of this encoder is defined by:

$$\mathbf{c} = ([\mathbf{c}\ \mathbf{b}] \otimes [\mathbf{c}\ \mathbf{b}])\,(\mathbf{G}^T \circ (\mathbf{M} \odot \mathbf{M}) + (\mathbf{L} \odot \mathbf{L}))$$

$$\mathbf{c} = \begin{bmatrix} c_1 & c_1 \\ c_1 & c_2 \\ c_1 & c_3 \\ c_1 & b_1 \\ c_1 & b_2 \\ c_1 & b_3 \\ \hline c_2 & c_1 \\ c_2 & c_2 \\ c_2 & c_3 \\ c_2 & b_1 \\ c_2 & b_2 \\ c_2 & b_3 \\ \hline c_3 & c_1 \\ c_3 & c_2 \\ c_3 & c_3 \\ c_3 & b_1 \\ c_3 & b_2 \\ c_3 & b_3 \\ \hline b_1 & c_1 \\ b_1 & c_2 \\ b_1 & c_3 \\ b_1 & b_1 \\ b_1 & b_2 \\ b_1 & b_3 \\ \hline b_2 & c_1 \\ b_2 & c_2 \\ b_2 & c_3 \\ b_2 & b_1 \\ b_2 & b_2 \\ b_2 & b_3 \\ \hline b_3 & c_1 \\ b_3 & c_2 \\ b_3 & c_3 \\ b_3 & b_1 \\ b_3 & b_2 \\ b_3 & b_3 \end{bmatrix}^T \left(\begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \\ \hline 0 & 0 & g_{31}^1 \\ 0 & 0 & g_{31}^2 \\ 0 & 0 & g_{31}^3 \\ 0 & 0 & g_{31}^4 \\ 0 & 0 & g_{31}^5 \\ 0 & 0 & g_{31}^6 \\ \hline 0 & g_{21}^1 & g_{32}^1 \\ 0 & g_{21}^2 & g_{32}^2 \\ 0 & g_{21}^3 & g_{32}^3 \\ 0 & g_{21}^4 & g_{32}^4 \\ 0 & g_{21}^5 & g_{32}^5 \\ 0 & g_{21}^6 & g_{32}^6 \\ \hline g_{11}^1 & g_{22}^1 & 0 \\ g_{11}^2 & g_{22}^2 & 0 \\ g_{11}^3 & g_{22}^3 & 0 \\ g_{11}^4 & g_{22}^4 & 0 \\ g_{11}^5 & g_{22}^5 & 0 \\ g_{11}^6 & g_{22}^6 & 0 \\ \hline g_{12}^1 & 0 & 0 \\ g_{12}^2 & 0 & 0 \\ g_{12}^3 & 0 & 0 \\ g_{12}^4 & 0 & 0 \\ g_{12}^5 & 0 & 0 \\ g_{12}^6 & 0 & 0 \\ \hline 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} \circ \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \\ \hline 0 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & 0 & 1 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \\ \hline 0 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & 1 & 1 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \\ \hline 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 1 & 0 \\ 1 & 1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 0 \\ \hline 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \\ 1 & 0 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 0 \\ \hline 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} + \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \\ \hline 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \\ \hline 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \\ \hline 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \\ \hline 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \\ \hline 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \\ 1 & 0 & 0 \end{bmatrix} \right)$$

Finally the system of equations that defines the encoder is:

$$\begin{bmatrix} c_3 \\ c_2 \\ c_1 \end{bmatrix} = \begin{cases} b_1 b_1 g_{11}^4 + b_1 b_2 g_{11}^5 + b_2 b_1 g_{12}^4 + b_2 b_2 g_{12}^5 + b_3 b_3 \\ c_3 c_3 g_{21}^3 + c_3 b_1 g_{21}^4 + b_1 c_3 g_{22}^3 + b_1 b_1 g_{22}^4 + b_2 b_2 \\ c_2 c_2 g_{31}^2 + c_2 c_3 g_{31}^3 + c_3 c_2 g_{32}^2 + c_3 c_3 g_{32}^3 + b_1 b_1 \end{cases}$$

Let a particular matrix $\mathbf{G}$ be such that this system of equations can be exactly defined as:

$$\begin{bmatrix} c_3 \\ c_2 \\ c_1 \end{bmatrix} = \begin{cases} b_1 + b_1 b_2 + b_3 \\ c_3 + b_2 \\ c_2 c_3 + b_1 \end{cases}$$

Let's suppose that the binary word $\mathbf{b}$=[1 1 1] is processed by the system. It's resulting word $\mathbf{c}$ would simply be:

$$\begin{array}{rclcrclcl} c_3 & = & b_1 + b_1 b_2 + b_3 & & c_3 & = & 1 + 1.1 + 1 & = & 1 \\ c_2 & = & c_3 + b_2 & \rightarrow & c_2 & = & 1 + 1 & = & 0 \\ c_1 & = & c_2 c_3 + b_1 & & c_1 & = & 0.1 + 1 & = & 1 \end{array}$$

Hence the word sent would be $\mathbf{c}$=[1 0 1].

The scheme of this particular encoder can be seen in figure (34). The AND gates that provide the non-linearities of the code. On the other hand, note that while the operation $b_1 + b_1 b_2$ is acceptable in the encoder due to its deterministic nature, it would need to be simplified to $b_1(1 + b_2)$ in the decoder, given that the first expression would produce a cycle in the decoding graph.

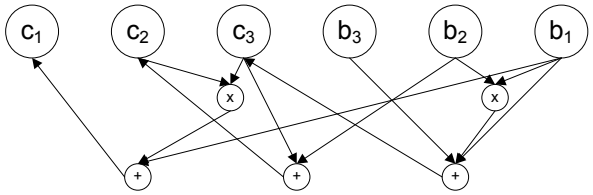

Figure 34: Factor-graph style diagram of the encoding process

7.3 Deterministic Pseudotriangular Recursive Linear Decoding

Let $\mathbf{c} = [c_1\ c_2\ c_3\ \ldots\ c_{N_b}] \in GF(2)^{1\times N_b}$ be the input, encoded bits, while the vector $\mathbf{b} = [b_1\ b_2\ b_3\ \ldots\ b_{N_b}] \in GF(2)^{1\times N_b}$ represents the decoded output bits of the stage.

$$\begin{aligned} \mathbf{D}_s \backslash \{\mathbf{G}_s \mathbf{D}_s = \mathbf{I}\} &\equiv \mathbf{D}_s = \mathbf{G}_s^{-1} \\ \mathbf{c} = \mathbf{b}\mathbf{G}_s &\rightarrow \mathbf{c}\mathbf{D}_s = \mathbf{b}\mathbf{G}_s\mathbf{D}_s \\ \mathbf{b}\mathbf{I} = \mathbf{c}\mathbf{D}_s &\rightarrow \mathbf{b} = \mathbf{c}\mathbf{D}_s \end{aligned} \tag{90}$$

As illustrated in (90), in order to reverse the function applied by a generic, non-specific linear code a inverse to the standard generator matrix $\mathbf{G}_s$, that is, the standard decoding

matrix $\mathbf{D}_s$, is needed. The problem is that generally speaking, as the number of bits to be processed rise ($N_b \uparrow$), the process of finding the inverse of $\mathbf{G}_s$ quickly gets computationally intensive - and even unrealistically high for large number of bits.

This issue can be avoided through the usage of the structure introduced in the pseudotriangular recursive linear encoder section. In the proposed implementation $\mathbf{G}$ is a $GF(2)^{N_b \times 2.N_b}$ matrix as depicted in (91): finding a inverse matrix to $\mathbf{G}$ won't be possible. However, finding a decoding expression with a $GF(2)^{N_b \times 2.N_b}$ decoding matrix $\mathbf{D}$ will.

$$\mathbf{G} = \begin{bmatrix} 0 & \cdots & 0 & 0 & g_{11} & \cdots & g_{1(N_b-2)} & g_{1(N_b-1)} & 1 \\ 0 & \cdots & 0 & g_{21} & g_{22} & \cdots & g_{2(N_b-1)} & 1 & 0 \\ 0 & \cdots & g_{31} & g_{32} & g_{33} & \cdots & 1 & 0 & 0 \\ \vdots & \ddots & \vdots & \vdots & \vdots & \ddots & \vdots & \vdots & \vdots \\ 0 & \cdots & g_{N_b(N_b-2)} & g_{N_b(N_b-1)} & 1 & \cdots & 0 & 0 & 0 \end{bmatrix} \tag{91}$$

The fastest strategy to analyze and express the decoder mathematical expressions is to start using the system of equations generated by the encoder, as shown in (78) and as a starting point, illustrated again in (92). Indeed, reversing the order of the equations in the decoder, and rearranging the variables withing each expression ends in a self-contained function that describes the decoder, depicted in (93).

$$\begin{bmatrix} c_{N_b} \\ c_{N_b-1} \\ c_{N_b-2} \\ \vdots \\ c_1 \end{bmatrix} = \begin{Bmatrix} b_1 g_{11} & + & b_2 g_{12} & + & \cdots & + & b_{N_b-1} g_{1(N_b-1)} & + & b_{N_b} \\ c_{N_b} g_{21} & + & b_1 g_{22} & + & \cdots & + & b_{N_b-2} g_{2(N_b-1)} & + & b_{N_b-1} \\ c_{N_b-1} g_{31} & + & c_{N_b} g_{32} & + & \cdots & + & b_{N_b-3} g_{3(N_b-1)} & + & b_{N_b-2} \\ \vdots & & \vdots & & \ddots & & \vdots & & \vdots \\ c_2 g_{N_b 1} & + & c_3 g_{N_b 2} & + & \cdots & + & c_{N_b-1} g_{N_b(N_b-1)} & + & b_1 \end{Bmatrix} \tag{92}$$

$$\downarrow$$

$$\begin{bmatrix} b_1 \\ \vdots \\ b_{N_b-2} \\ b_{N_b-1} \\ b_{N_b} \end{bmatrix} = \begin{Bmatrix} c_1 & + & c_2 g_{N_b 1} & + & c_3 g_{N_b 2} & + & \cdots & + & c_{N_b-1} g_{N_b(N_b-1)} \\ \vdots & & \vdots & & \vdots & & \ddots & & \vdots \\ c_{N_b-2} & + & c_{N_b-1} g_{31} & + & c_{N_b} g_{32} & + & \cdots & + & b_{N_b-3} g_{3(N_b-1)} \\ c_{N_b-1} & + & c_{N_b} g_{21} & + & b_1 g_{22} & + & \cdots & + & b_{N_b-2} g_{2(N_b-1)} \\ c_{N_b} & + & b_1 g_{11} & + & b_2 g_{12} & + & \cdots & + & b_{N_b-1} g_{1(N_b-1)} \end{Bmatrix} \tag{93}$$

Note that the system of equations in (93) is enough to efficiently decode any word received. However, in order to properly define the decoder, and relate it to the vectorial expressions that describe the encoder in (77), the decoding matrix $\mathbf{D}$ has to be ascertained.

The expression (93) can be used to determine the decoding matrix $\mathbf{D}$. In particular, the decoding matrix $\mathbf{D}$ itself, shown in (94), can be obtained by examining the location of each $g_{ij}, i, j = 1, 2, \ldots, N_b - 1$ in the expressions placed at the right of the system of equations of the decoder (93)

$$\mathbf{D} = \begin{bmatrix} 1 & \cdots & g_{N_b(N_b-2)} & g_{N_b(N_b-1)} & 0 & \cdots & 0 & 0 & 0 \\ \vdots & \ddots & \vdots & \vdots & \vdots & \ddots & \vdots & \vdots & \vdots \\ 0 & \cdots & g_{31} & g_{32} & g_{33} & \cdots & 0 & 0 & 0 \\ 0 & \cdots & 1 & g_{21} & g_{22} & \cdots & g_{2(N_b-1)} & 0 & 0 \\ 0 & \cdots & 0 & 1 & g_{11} & \cdots & g_{1(N_b-2)} & g_{1(N_b-1)} & 0 \end{bmatrix} \tag{94}$$

$$\mathbf{D} = [\mathbf{I}\ \mathbf{I}] + \begin{bmatrix} 0 & \cdots & 0 & 1 \\ 0 & \cdots & 1 & 0 \\ \vdots & \ddots & \vdots & \vdots \\ 1 & \cdots & 0 & 0 \end{bmatrix} \mathbf{G} \tag{95}$$

The relationship between the matrices $\mathbf{G}$ and $\mathbf{D}$ can be seen in (95). $[\mathbf{I}\ \mathbf{I}]$ are two concatenaded $N_b \times N_b$ identity matrices that create a $N_b \times 2.N_b$ matrix. The simple and immediate relationship between $\mathbf{D}$ and $\mathbf{G}$ allows instant knowledge of the decoding matrix once the enconding matrix is known, avoiding costly calculus as in the case of generic linear codes.

Finally, knowing $\mathbf{D}$ and using as template the system of equations of the decoder in (93) the expression that mathematically defines the linear decoder of the proposed implementation can be illustrated with either of the following expressions:

$$\mathbf{b}^T = \mathbf{D}[\mathbf{c}\ \mathbf{b}]^T \tag{96}$$

$$\mathbf{b}^T = \left([\mathbf{I}\ \mathbf{I}] + \begin{bmatrix} 0 & \cdots & 0 & 1 \\ 0 & \cdots & 1 & 0 \\ \vdots & \ddots & \vdots & \vdots \\ 1 & \cdots & 0 & 0 \end{bmatrix} \mathbf{G} \right) [\mathbf{c}\ \mathbf{b}]^T \tag{97}$$

Example: In the proposed linear encoder section example (7.1) the $\mathbf{b} = [1\ 1\ 1\ 1\ 1\ 1\ 1\ 0]$ word was processed and the $\mathbf{c} = [1\ 0\ 0\ 0\ 0\ 0\ 0\ 0]$ vector was obtained with a encoder with the following $\mathbf{G}$ matrix:

$$\mathbf{G} = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

The task is to decode the received **c** word so **b** can be retrieved.

The fastest, most straightforward way of decoding the received word **c** would be to directly use the system of equations of the decoder shown in (93). However, an alternative, interesting path will be chosen in this example.

By using the expression (95) we can find the decoding matrix **D** is defined as:

$$\mathbf{D} = \left[\begin{array}{cccccccc:cccccccc} 1 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \end{array}\right]$$

The division in the matrix has been used to emphasize the fact that the columns of the left define, each one, the usage or non-usage of each of the elements of the word **c** ($c_1\ c_2\ \ldots\ c_{N_b}$), while the columns at the right do the same function for the elements of **b** ($b_1\ b_2\ \ldots\ b_{N_b}$). For instance, the '1' in the first row and first column means that, for the first decoded bit (b_1), the binary element c_1 will be part of the sum of $GF(2)$ elements. Consequently, the decoding system of equations of this example, and by substitution, decoded **b** word would be:

$$\mathbf{c} = [1\ 0\ 0\ 0\ 0\ 0\ 0\ 0]$$

$$\begin{array}{rcl} b_1 &=& c_1 + c_2 + c_8 \\ b_2 &=& c_2 + c_3 + b_1 \\ b_3 &=& c_3 + c_4 + b_2 \\ b_4 &=& c_4 + c_5 + b_3 \\ b_5 &=& c_5 + c_6 + b_4 \\ b_6 &=& c_6 + c_7 + b_5 \\ b_7 &=& c_7 + c_8 + b_6 \\ b_8 &=& c_8 + b_1 + b_7 \end{array} \rightarrow \begin{array}{rcl} b_1 &=& 1+0+0 = 1 \\ b_2 &=& 0+0+1 = 1 \\ b_3 &=& 0+0+1 = 1 \\ b_4 &=& 0+0+1 = 1 \\ b_5 &=& 0+0+1 = 1 \\ b_6 &=& 0+0+1 = 1 \\ b_7 &=& 0+0+1 = 1 \\ b_8 &=& 0+1+1 = 0 \end{array}$$

Thereby recovering the originally sent vector **b** =[1 1 1 1 1 1 1 0], used in the proposed linear encoder section (7.1) to code the word **c** =[1 0 0 0 0 0 0 0].

7.4 Deterministic Pseudotriangular Recursive Non-Linear Decoding

Finding an inverse decoding matrix ($\mathbf{D}_s$) to a generic non-linear encoder with a generator matrix ($\mathbf{G}_s$) such as the one depicted in (7.2.1) isn't feasible. $\mathbf{G}_s$ itself is not a square matrix, and finding an equivalent decoding expression can be a cumbersome, time consuming task.

Fortunately, the proposed implementation has, as in the linear case, a simple and fast method for finding the decoding expressions given the $GF(2)^{N_b \times 2.N_b^D}$ encoding matrix $\mathbf{G}$. Such matrix is, indeed, exactly the same one used in the encoding process:

$$\mathbf{G} = \begin{bmatrix} \mathbf{0} & \cdots & \mathbf{0} & \mathbf{0} & \mathbf{g}_{11} & \cdots & \mathbf{g}_{1(N_b-2)} & \mathbf{g}_{1(N_b-1)} & \mathbf{0} \\ \mathbf{0} & \cdots & \mathbf{0} & \mathbf{g}_{21} & \mathbf{g}_{22} & \cdots & \mathbf{g}_{2(N_b-1)} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \cdots & \mathbf{g}_{31} & \mathbf{g}_{32} & \mathbf{g}_{33} & \cdots & \mathbf{0} & \mathbf{0} & \mathbf{0} \\ \vdots & \ddots & \vdots & \vdots & \vdots & \ddots & \vdots & \vdots & \vdots \\ \mathbf{0} & \cdots & \mathbf{g}_{N_b(N_b-2)} & \mathbf{g}_{N_b(N_b-1)} & \mathbf{0} & \cdots & \mathbf{0} & \mathbf{0} & \mathbf{0} \end{bmatrix} \tag{98}$$

Remember that $\mathbf{g}_{ij}$ are the $1 \times (2N_b)^{D-1}$ vectors containing the operations that will be done by the non-linear encoder, while the bold $\mathbf{0}$ are equally sized vectors whose elements are all 0.

Once again, let, $\mathbf{c} = [c_1\ c_2\ c_3\ \ldots\ c_{N_b}] \in GF(2)^{1\times N_b}$ be the input vector, the encoded bits, while the vector $\mathbf{b} = [b_1\ b_2\ b_3\ \ldots\ b_{N_b}] \in GF(2)^{1\times N_b}$ represents the decoded output bits of the stage. As in the linear case, the fastest, most straightforward way for finding with such expression is examining the system of equations of the encoder, shown in (89) and also, as reference, in (99). By merely reversing the order of the equations, and rearranging the variables within them, the system of equations of the decoder can be obtained.

$$\begin{bmatrix} c_{N_b} \\ c_{N_b-1} \\ c_{N_b-2} \\ \vdots \\ c_1 \end{bmatrix} = \begin{cases} b_1 g_{11}^{N_b+1} + b_1 b_2 g_{11}^{N_b+2} + \cdots + b_{N_b-1} b_{N_b-1} g_{1(N_b-1)}^{(2N_b)^{D-1}-1} + b_{N_b} \\ c_{N_b} g_{21}^{N_b} + c_{N_b} b_1 g_{21}^{N_b+1} + \cdots + b_{N_b-2} b_{N_b-2} g_{2(N_b-1)}^{(2N_b)^{D-1}-2} + b_{N_b-1} \\ c_{N_b-1} g_{31}^{N_b-1} + c_{N_b-1} c_{N_b} g_{31}^{N_b} + \cdots + b_{N_b-3} b_{N_b-3} g_{3(N_b-1)}^{(2N_b)^{D-1}-3} + b_{N_b-2} \\ \vdots \qquad\qquad \vdots \qquad\qquad \ddots \qquad\qquad \vdots \qquad\qquad \vdots \\ c_2 g_{N_b 1}^2 + c_2 c_3 g_{N_b 1}^3 + \cdots + c_{N_b} c_{N_b} g_{N_b(N_b-1)}^{(2N_b)^{D-1}-N_b} + b_1 \end{cases} \tag{99}$$

$$\downarrow$$

$$\begin{bmatrix} b_1 \\ \vdots \\ b_{N_b-2} \\ b_{N_b-1} \\ b_{N_b} \end{bmatrix} = \begin{cases} c_1 + c_2 g_{N_b 1}^2 + c_2 c_3 g_{N_b 1}^3 + \cdots + c_{N_b} c_{N_b} g_{N_b(N_b-1)}^{(2N_b)^{D-1}-N_b} \\ \vdots \quad\quad \vdots \quad\quad \vdots \quad\quad \ddots \quad\quad \vdots \\ c_{N_b-2} + c_{N_b-1} g_{31}^{N_b-1} + c_{N_b-1} c_{N_b} g_{31}^{N_b} + \cdots + b_{N_b-3} b_{N_b-3} g_{3(N_b-1)}^{(2N_b)^{D-1}-3} \\ c_{N_b-1} + c_{N_b} g_{21}^{N_b} + c_{N_b} b_1 g_{21}^{N_b+1} + \cdots + b_{N_b-2} b_{N_b-2} g_{2(N_b-1)}^{(2N_b)^{D-1}-2} \\ c_{N_b} + b_1 g_{11}^{N_b+1} + b_1 b_2 g_{11}^{N_b+2} + \cdots + b_{N_b-1} b_{N_b-1} g_{1(N_b-1)}^{(2N_b)^{D-1}-1} \end{cases} \tag{100}$$

Remember that the notation g_{ij}^k is entirely equivalent to the more orthodox $g_{ij}(k)$: g_{ij}^k represents the k-th element of the $\mathbf{g}_{ij}$ vector. The expression in (100) is sufficient to decode any $\mathbf{c}$ word into the original $\mathbf{b}$ vector. However, as in the preceding linear case, the vectorial equation that defines the non-linear decoding system will be also illustrated in this section.

Using the entire system of equations of the decoder as a template ((100) is a brief summary of the entire expression), it is easy to notice that the equivalent to the matrix $\mathbf{M}$ of the encoder, $\mathbf{M}_i$ remains equal but for a reverse in the row order. The equivalent to the matrix $\mathbf{L}$, $\mathbf{L}_i$, changes substantially to accommodate for the fact that now it will be c_1 to c_{N_b} the elements that will be used to find b_1 to b_{N_b}, unlike in the encoder:

$$\mathbf{M}_i = \left[\begin{array}{cccccccccccc} 0 & 1 & 1 & \cdots & 1 & 1 & 0 & 0 & \cdots & 0 & 0 & 0 \\ \hline 0 & 0 & 1 & \cdots & 1 & 1 & 1 & 0 & \cdots & 0 & 0 & 0 \\ \hline \vdots & \vdots & \vdots & \ddots & \vdots & \vdots & \vdots & \vdots & \ddots & \vdots & \vdots & \vdots \\ \hline 0 & 0 & 0 & \cdots & 1 & 1 & 1 & 1 & \cdots & 0 & 0 & 0 \\ \hline 0 & 0 & 0 & \cdots & 0 & 1 & 1 & 1 & \cdots & 1 & 0 & 0 \\ \hline 0 & 0 & 0 & \cdots & 0 & 0 & 1 & 1 & \cdots & 1 & 1 & 0 \end{array}\right]^T = \left(\begin{bmatrix} 0 & \cdots & 0 & 1 \\ 0 & \cdots & 1 & 0 \\ \vdots & \ddots & \vdots & \vdots \\ 1 & \cdots & 0 & 0 \end{bmatrix} \mathbf{M} \right)^T$$

$$\mathbf{L}_i = \left[\begin{array}{cccccccccccc} 1 & 0 & 0 & \cdots & 0 & 0 & 0 & 0 & \cdots & 0 & 0 & 0 \\ \hline 0 & 1 & 0 & \cdots & 0 & 0 & 0 & 0 & \cdots & 0 & 0 & 0 \\ \hline 0 & 0 & 1 & \cdots & 0 & 0 & 0 & 0 & \cdots & 0 & 0 & 0 \\ \hline \vdots & \vdots & \vdots & \ddots & \vdots & \vdots & \vdots & \vdots & \ddots & \vdots & \vdots & \vdots \\ \hline 0 & 0 & 0 & \cdots & 1 & 0 & 0 & 0 & \cdots & 0 & 0 & 0 \\ \hline 0 & 0 & 0 & \cdots & 0 & 1 & 0 & 0 & \cdots & 0 & 0 & 0 \end{array}\right]^T = \left([\mathbf{I}\ \mathbf{I}] + \begin{bmatrix} 0 & \cdots & 0 & 1 \\ 0 & \cdots & 1 & 0 \\ \vdots & \ddots & \vdots & \vdots \\ 1 & \cdots & 0 & 0 \end{bmatrix} \mathbf{L} \right)^T \tag{101}$$

Finally, the expression that produces the system of equations of the decoder turns out to be very similar to the encoder one:

$$\mathbf{b} = \underbrace{([\mathbf{c}\ \mathbf{b}] \otimes [\mathbf{c}\ \mathbf{b}] \otimes \ldots \otimes [\mathbf{c}\ \mathbf{b}])}_{D-1\ \otimes\ \text{operations}} (\mathbf{D}^T \circ \underbrace{(\mathbf{M}_i \odot \mathbf{M}_i \odot \ldots \odot \mathbf{M}_i)}_{D-1\ \odot\ \text{operations}} + \underbrace{(\mathbf{L}_i \odot \mathbf{L}_i \odot\ \ldots\ \odot \mathbf{L}_i)}_{D-1\ \odot\ \text{operations}}) \tag{102}$$

Where the operations $\otimes$, $\odot$ and $\circ$ are the Kronecker, Khatri-Rao and Hadamard products, depicted in (83),(86) and (87) respectively. On the other hand, $\mathbf{D}$ is the decoding matrix. In this implementation, it is equivalent to a row reversed order $\mathbf{G}$ matrix:

$$\begin{aligned}\mathbf{D} &= \begin{bmatrix} \mathbf{0} & \cdots & \mathbf{g}_{N_b(N_b-2)} & \mathbf{g}_{N_b(N_b-1)} & \mathbf{0} & \cdots & \mathbf{0} & \mathbf{0} & \mathbf{0} \\ \vdots & \ddots & \vdots & \vdots & \vdots & \ddots & \vdots & \vdots & \vdots \\ \mathbf{0} & \cdots & \mathbf{g}_{31} & \mathbf{g}_{32} & \mathbf{g}_{33} & \cdots & \mathbf{0} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \cdots & \mathbf{0} & \mathbf{g}_{21} & \mathbf{g}_{22} & \cdots & \mathbf{g}_{2(N_b-1)} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \cdots & \mathbf{0} & \mathbf{0} & \mathbf{g}_{11} & \cdots & \mathbf{g}_{1(N_b-2)} & \mathbf{g}_{1(N_b-1)} & \mathbf{0} \end{bmatrix} \\ &= \begin{bmatrix} 0 & \cdots & 0 & 1 \\ 0 & \cdots & 1 & 0 \\ \vdots & \ddots & \vdots & \vdots \\ 1 & \cdots & 0 & 0 \end{bmatrix} \mathbf{G} \end{aligned} \tag{103}$$

Example: In the example of section (7.2) a word $\mathbf{b}$=[1 1 1] was sent through a non-linear encoder with the following system of equations:

$$\begin{bmatrix} c_3 \\ c_2 \\ c_1 \end{bmatrix} = \begin{cases} b_1 + b_1 b_2 + b_3 \\ c_3 + b_2 \\ c_2 c_3 + b_1 \end{cases}$$

The word that was sent was $\mathbf{c}$=[1 0 1]. This section will demonstrate the correct reversal of such encoding.

In order to decode the word, the fastest method is to reverse in order, and rearrange the terms of the system of equations of the encoder so it's expressed properly for the decoding to take place, as shown in this section. Later, the encoded word. In other words:

$$\begin{array}{lcl} c_3 = b_1 + b_1 b_2 + b_3 \\ c_2 = c_3 + b_2 \\ c_1 = c_2 c_3 + b_1 \end{array} \rightarrow \begin{array}{lcl} b_1 = c_1 + c_2 c_3 \\ b_2 = c_2 + c_3 \\ b_3 = c_3 + b_1 + b_1 b_2 \end{array} \rightarrow \begin{array}{lcll} b_1 = 1 + 0.1 & = & 1 \\ b_2 = 0 + 1 & = & 1 \\ b_3 = 1 + 1 + 1.1 & = & 1 \end{array}$$

Hence recovering the word initially sent, $\mathbf{b}$=[1 1 1].

References

[1] Meritxell Lamarca, Hanqing Lou and Javier Garcia-Frias. *Random Labeling: A New Approach to Achieve Capacity in MIMO Quasi-Static Fading Channels*. IEEE International Symposium on Information Theory (ISIT), 2006.

[2] David Tse, Pramod Viswanath. *Fundamentals of Wireless Communication*. Cambridge University Press, 2004.

[3] Iterative Techniques for Optimal Detection in Coded Communications (TIDOC) documentation. Technical University of Catalonia, 2005.

[4] Eduardo Zacarías B. *BLAST Architectures* Postgraduate Course in Radiocomunications, 2004.

[5] Hesham El Gamal, A. Roger Hammons. *A New Approach to Layered Space-Time Coding and Signal Processing*. IEEE Transactions on information theory, vol. 47, No. 6, 2001.

[6] Robert G. Gallager. *Low Density Parity Check Codes* Monograph, M.I.T. Press, 1963.

[7] Stephan ten Brink, *Convergence Behavior of Iteratively Decoded Parallel Concatenated Codes* IEEE Transactions on communications, 2001.

[8] Udo Wachsmann, Member, IEEE, Robert F. H. Fischer, Member, IEEE, and Johannes B. Huber, Member, IEEE *Multilevel Codes: Theoretical Concepts and Practical Design Rules* IEEE Transactions on information theory, vol. 45, No. 5, July 1999

[9] Anonymous. *Mutual Information and Channel Capacity*. Online: http://www.cmlab.csie.ntu.edu.tw/ ipr/mmsec2011/data/lecture/Lecture5

[10] Wikipedia. *Wikipedia Article: MIMO*. https://en.wikipedia.org/wiki/MIMO

[11] ETSI. *Digital Video Broadcasting (DVB); Second generation framing structure, channel coding and modulation systems for Broadcasting, Interactive Services, News Gathering and other broadband satellite applications (DVB-S2)*. Draft ETSI EN 302 307 V1.3.1 (2012-11)

[12] Xiaodong Wang, Member, IEEE, and H. Vincent Poor, Fellow, IEEE. *Iterative (Turbo) Soft Interference Cancellation and Decoding for Coded CDMA* IEEE Transactions on Communications, vol. 47, No. 7, July 1999